LA-UR-07-8214

CartaBlanca User's Manual

by

P.T. Giguere, X. Ma, N.T. Padial-Collins, D.Z. Zhang, and Q. Zou

Los Alamos National Laboratory
Los Alamos, NM

W.B. VanderHeyden

BP Corporation
Naperville, IL

December 2007

ABSTRACT

CartaBlanca is an object-oriented nonlinear simulation and prototyping software package whose main functions are to assist both analysts and code developers in solving a wide range of hydrodynamics and fluid/structure-interaction problems.

The CartaBlanca User's Guide provides comprehensive instruction on the use of CartaBlanca to obtain and analyze results for the broad range of problem domains the code is applicable to. The User's Guide includes a description of CartaBlanca's capabilities, a "quick start" to using the code, complete input specifications (including description of a graphical user interface that assists in preparing input files), and sections on the running of CartaBlanca, modeling guidelines, and the code's output files and printouts.

This manual is one of three documents that comprise the main CartaBlanca documentation set. The other two are the Theory Manual [12] and the Programmer's Manual [13].

CONTENTS

FIGURES

TABLES

1. INTRODUCTION

This document provides a comprehensive guide to the use of the CartaBlanca computer program to obtain and analyze results for the broad range of problem domains in hydrodynamics and fluid-structure interaction for which the code is applicable. An overview of CartaBlanca's capabilities is given in Section 2, where the reader is also directed to the CartaBlanca website for additional information.

Section 3 gives a "quick start" to using CartaBlanca. Complete input specifications are given in Section 4; in addition, Section 4 describes a graphical user interface that has been developed to assist in preparing input files. Section 5 describes the running of CartaBlanca, including the platforms supported. The form of CartaBlanca's output files and printouts, and their analysis, are discussed in Section 6. Guidelines for use of the code's many input options and features are given in Section 7.

This manual is one of three documents that comprise the main CartaBlanca documentation set. The other two are the CartaBlanca Theory Manual [12] and the CartaBlanca Programmer's Manual [13]. The Theory Manual gives a detailed description of the code's physics and numerical basis, including the governing conservation equations, their closure models and discretization, available constitutive models, and the numerical solution methods. The Programmer's Manual describes the code's structure, computational flow, and database; it references relevant sections of the Theory Manual.

2. CartaBlanca OVERVIEW

CartaBlanca is an object-oriented component-based simulation and prototyping software package that enables both analysts and code developers to solve a wide range of nonlinear hydrodynamics and fluid/structure-interaction problems on unstructured grids and graphs. Although the user of CartaBlanca does not need to know the details of the code's implementation, she or he should be aware that CartaBlanca was designed to be readily extendable to new physical models. CartaBlanca is written entirely in Java; therefore it provides scientists and engineers with developer-friendly, modular software to use in producing large-scale computational models. CartaBlanca allows users to solve a wide variety of nonlinear physics problems, including multiphase flows, interfacial flows, solidifying flows, and complex material responses. CartaBlanca makes use of the powerful, state-of-the-art Jacobian-free Newton-Krylov method to solve nonlinear equations in a flexible unstructured grid finite-volume scheme. CartaBlanca couples a Material Point Method (MPM) implementation of the Particle-in-Cell (PIC) method (a technique used to model discrete objects), with its Arbitrary Lagrangian Eulerian (ALE) multiphase flow treatment, to model fluid interaction with solid materials that can undergo deformation, damage, and failure. The MPM/PIC method can also be used to model solid-solid interactions.

Calculations can be run in 1-D, 2-D, or 3-D on a wide variety of unstructured grids with triangular, quadrilateral, tetrahedral, and hexahedral elements. This design allows CartaBlanca to handle complex geometrical shapes and mathematical domains. Cartesian, cylindrical, or spherical coordinates can be used.

Because CartaBlanca is written entirely in Java, it is highly portable and readily installed on any platform with a Java runtime environment available. CartaBlanca has been run on platforms ranging from Windows laptops to supercomputers. Runtime performance is close to that of Fortran hydrodynamics codes. Parallel computation is built into the code: CartaBlanca is designed around Java's built-in multi-thread capability, where processes can be run simultaneously and can communicate with each other, but are controlled from the same program. Both shared and distributed memory architectures are supported.

The preparation of input files is greatly facilitated by a graphical user interface that is provided with the code. Also, an extensive set of test problems is provided; these problems can be used as templates for the creation of other input models.

Output is written to text files in both Tecplot [11] format and ParaView [9] format (Note: The ParaView capability is currently under development).

2.1. CartaBlanca Website

A good introduction to CartaBlanca's motivation, design, and capabilities can be found at the CartaBlanca website:

    http://www.lanl.gov/projects/CartaBlanca/

3. CartaBlanca QUICK START

Here we provide "quick start" guidance on installing CartaBlanca, the code's input requirements, running the code, the CartaBlanca test suite, creating a new problem, and viewing the output.

CartaBlanca is very easy to install and run. We provide scripts to compile and run the code from the Unix command line (which can be easily modified for Windows/DOS). Alternatively, the user may wish to use one of the integrated development environments (IDEs) for Java (NetBeans, Eclipse, JBuilder, etc.), which are available at no cost on the Internet.

3.1. Computer Platforms and Installation

CartaBlanca is distributed as a single `.zip` file that contains the executable code, source code, scripts for building and running the code (we describe building and running CartaBlanca below), an extensive set of sample input-specification files that spans a wide range of applications, and documentation.

Functionally, the code is comprised of four elements: (1) the solution engine ("main code") that reads and processes input (problem specification) files and writes the output, (2) a graphical user interface (GUI) that assists the user in preparing an input-specification, (3) a set of routines ("methods" in Java parlance) that sets up and drives an extensive test suite for the code that is based on the Java JUnit facility, and (4) a set of Java methods that can be used to generate mesh files that specify a problem domain. While these four code elements are functionally distinct, the CartaBlanca software is written and organized as a single integrated set of program source files; essentially, different entry points are specified at run time to select the desired functionality (much of the code is also shared). A large set of mesh files that specify computational grids in 1-D, 2-D, and 3-D is included in the distribution. The distribution also includes Unix scripts to build and run CartaBlanca, JBuilder projects to do the same (see below), and an XML file to build the code with the `ant` utility.

Because CartaBlanca is written entirely in Java, it can be run on any computer platform with a Java runtime environment (e.g., Unix/Linux/Solaris, Windows, Mac OS). The CartaBlanca package itself is less than 400 MB; all the test cases in the distribution package have been run with the Java parameter `–mx64m` (maximum memory of 64MB). The code has been run on platforms ranging from laptops to supercomputer clusters. Currently at Los Alamos, Java versions 1.4.n are being used for CartaBlanca development and applications.

Java is available at no cost at

`http://java.sun.com/` (Windows, Linux, Solaris).

For the Macintosh, Java is bundled with Mac OS X.

We recommend that a complete Java software development kit (SDK) be obtained, to allow both code execution and compilation (e.g., for compiling the JUnit test suite drivers).

CartaBlanca's main output is in a text file format that is compatible with the commercial Tecplot package [11]; this format is readily adaptable to other graphics software. Optionally, the user may select output in the format read by the free ParaView package [9] (Note: The ParaView capability is currently under development).

As discussed in the following section, the user may wish to run CartaBlanca with one the integrated development environments (IDEs) for Java (NetBeans, Eclipse, JBuilder, Idea, etc.), some of which are available at no cost. NetBeans is available at no cost at

`http://www.netbeans.org/` (Windows, Linux, Solaris, Mac OS X)

A convenient bundle of Java and NetBeans is available at no cost at

`http://java.sun.com/` (Windows, Linux, Solaris).

A basic version of JBuilder (entirely adequate for CartaBlanca) is available at no cost at

http://www.borland.com/downloads/download_jbuilder.html (Windows, Linux, Solaris, Mac OS) (download Foundation 2005 version).

We run the CartaBlanca test suite with JUnit [5], which is also available at no cost. We include a JUnit executable in the distribution. Currently we typically run JUnit either as a JBuilder project (the project file is included in the distribution), or from the Unix command line; JUnit is also bundled with NetBeans and Eclipse.

3.1.1. Overview of Release Package

CartaBlanca is distributed as a self-contained `.zip` file, which contains a top-level directory with several individual files, and a number of sub-directories (which in turn can have sub-directories). All supported platforms (i.e., Java-enabled) can use this `.zip` file. The top-level directory is called `cartablanca`; it contains a number of useful support files that provide a quick means to get CartaBlanca running, including

- JBuilder projects (suffix `.jpr`) that compile and run the GUI, the JUnit test suite, and the main code: `rungui.jpr`, `cbtests.jpr`, and `cbphysmain.jpr`.
- Windows `.cmd` file to run the GUI: `rungui.cmd`.
- `.xml` file for the user who wishes to use the `ant` utility to build the code: `build.xml`.
- Unix scripts for building and running the GUI, test suite, and main code, in directory `scripts/unix`.

The rest of this quick start makes use of files in the following directories:

- `src`: the complete set of CartaBlanca source code files, organized according to CartaBlanca's Java `package` hierarchy.
- `testIO`: CartaBlanca input-specifier files that are generated by running the test suite.
- `meshes`: files that specify a large set of sample computational grids in 1D, 2D, and 3D.
- `output`: graphics output files and binary restart dumps from a calculation.

There are many other directories and files in the distribution that support the code or help the CartaBlanca user, including the documentation set, reports, and sample graphics stylesheets and macros. An overview of the directories and files in the distribution `.zip` file is given in Appendix B.

3.2. Input Files

The numerical and physical specifications that define a problem are contained in a text file that is named, by default,

```
            InputSpecifier.IO
```

File `InputSpecifier.IO` is used to specify, e.g., time-step controls, files containing the computational grid, physics packages to be solved, solution algorithms, initial and boundary conditions, and material properties. Guidance on quickly preparing an `InputSpecifier.IO` is given below in Sections 3.4 ("Test Suite") and 3.5 ("Sample Project"). Complete specifications are given in Section 4 ("Input Preparation and Specifications").

In addition to file `InputSpecifier.IO`, CartaBlanca can read six additional files that are called collectively a problem's Mesh Input Files; two of these are required and four are optional. The two required files specify the problem domain's computational node locations and mesh (node) connectivity. A third file is required to specify mesh partitions for parallel runs. Three additional files can be provided at the user's option. The six Mesh Input Files are:

`NodeDataFile`, node coordinates (required),
`MeshFile`, the mesh connectivity (required),
`MeshPartitionFile`, mesh partitioning (required for parallel runs),
`ParticleFile`, particle-model data (optional, an automatic calculation can be chosen),
`BoundaryFile`, boundary conditions (optional, can be given in `InputSpecifier.IO`), and
`InitialConditionsFile`, initial conditions (optional, can be given in `InputSpecifier.IO`)

These file names are not required; each of the Mesh Input Files can be named according to the user's wishes. All are text files. `MeshFile`, `NodeDataFile`, and `MeshPartitionFile` are in the format of the METIS mesh-partitioning code [6]. Creation and use of a simple `MeshFile` and `NodeDataFile` are described in Section 3.5 ("Sample Project"). Input of the six Mesh Input Files to a CartaBlanca calculation is described in Section 4.1 ("General Information"), and complete specifications are given in Section 4.1.1 ("Mesh Input Files").

3.3. Running CartaBlanca

CartaBlanca can be compiled and run from the Unix/Linux or Windows (DOS) command line, or with a button-click from a Java IDE (NetBeans, Eclipse, JBuilder, Idea, etc.).

To compile and run from the Unix command line (assuming that the CartaBlanca package has been installed in `~myhome/cartablanca`, and that Java is installed):

- Set an environment variable `CBROOT`, e.g. in a C shell:

  ```
  setenv CBROOT ~myhome/cartablanca
  ```

- Go to `~myhome/cartablanca`:

  If the program `ant` is installed in the system, enter:

  ```
  >ant
  ```

  to compile the package (where > is the system prompt).

  If `ant` is not installed, enter

  ```
  >cp scripts/unix/* .
  ```

  to copy all the Unix scripts under the directory `cartablanca`, then, enter

  ```
  >compPhysMain.unix

  >compPhysTests.unix
  ```

to compile the main code and the test code, respectively.

- Now the Unix script `runPhysTests.unix` can be used to run the CartaBlanca test suite, and the script `runPhysMain.unix` to run the main code for a specific problem. Section 3.2 described the basic input-file requirements. Sections 3.4 and 3.5 show how to obtain a set of sample input files (by running the test suite), and how to customize an input problem.

Los Alamos has made extensive use of the Borland JBuilder IDE. The distribution package includes files `cartablanca.jpr`, `cbphysmain.jpr`, `rungui.jpr`, and `cbtests.jpr` in directory `~myhome/cartablanca`. These are JBuilder project files that can be used to build and run the main code, the GUI, and the test suite (the short-running problems, see Section 3.4). If another IDE is used, specify the following targets:

`gov.lanl.cartablanca.main.PhysMain` (main code for a specific problem)
`gov.lanl.cartablanca.main.RunGUI(GUI)`
`gov.lanl.cartablanca.test.AllTests(`test suite's short-running problems).

3.4. Test Suite

The CartaBlanca distribution package includes a test suite that is run by developers to check code modifications. There are 47 standard short-running problems that have been developed to check many aspects of the code's logic, insuring that code changes do not have unintended effects. These tests are run using JUnit, which is available at no cost [5], and is included in the CartaBlanca distribution and in many IDE packages (e.g., JBuilder, NetBeans, etc.). The entire short-running test suite is set up and run by executing a single CartaBlanca Java method (see `runPhysTests.unix`, or the target of `cbtests.jpr`); typically the 47 problems run in 1 – 2 minutes on a desktop computer.

We recommend that the test suite be run to obtain an introduction to CartaBlanca and to generate a set of sample input problems. The test logic automatically writes 47 "`.IO`" files in the format of an `inputSpecifier.IO` file, to directory `testIO`; it then proceeds to execute these files. Success or failure of a given problem's results is specified in the CartaBlanca source code, according to JUnit protocols, and is automatically tested by JUnit. All required mesh files are in the distribution, in subdirectories under directory `meshes`.

On the Unix command line, enter

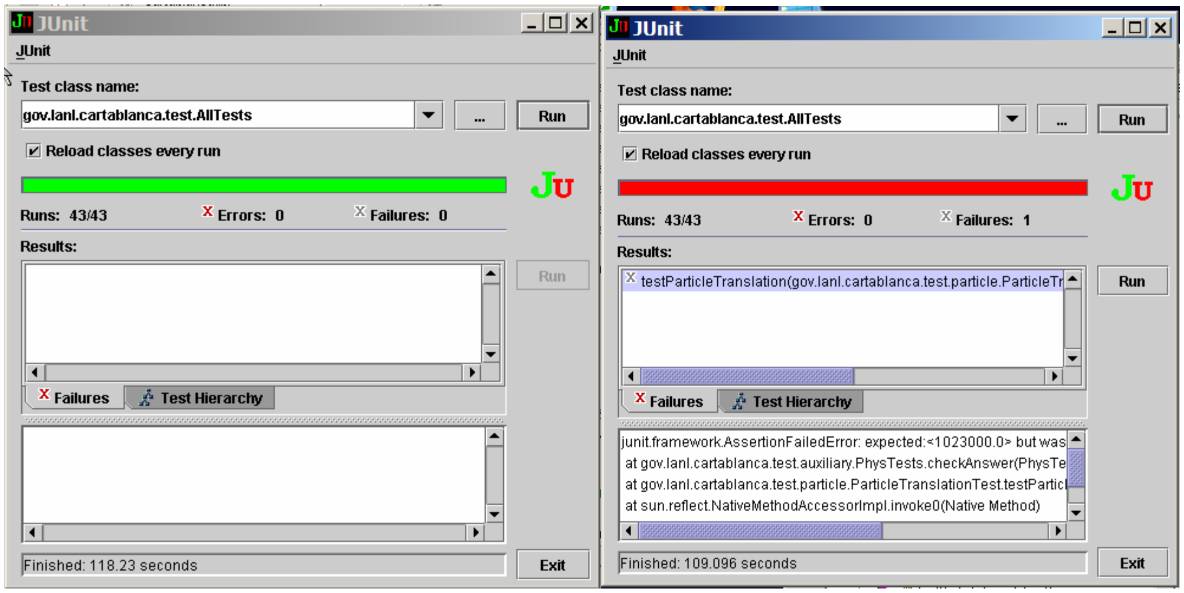
```
>runPhysTests.unix
```

Or, if an IDE is used, run

```
gov.lanl.cartablanca.test.AllTests
```

(this is the target of `cbtests.jpr`).

A JUnit window will show the test status as the problems automatically execute. The two displays in Figure 1 show success and failure.



Test class name: gov.lanl.cartablanca.test.AllTests

☑ Reload classes every run

Runs: 43/43 Errors: 0 Failures: 0

Results:

Finished: 118.23 seconds

Test class name: gov.lanl.cartablanca.test.AllTests

☑ Reload classes every run

Runs: 43/43 Errors: 0 Failures: 1

Results:

```
testParticleTranslation(gov.lanl.cartablanca.test.particle.ParticleTr
```

```
junit.framework.AssertionFailedError: expected:<1023000.0> but was
 at gov.lanl.cartablanca.test.auxiliary.PhysTests.checkAnswer(PhysTe
 at gov.lanl.cartablanca.test.particle.ParticleTranslationTest.testParticl
 at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

Finished: 109.096 seconds

Figure 1. JUnit test displays.

Of course, a CartaBlanca distribution should run the test suite successfully.

In addition to the 47 short-running test problems, there are five “longer-running” problems that we typically run with a Unix script; the code that generates these problems, and their mesh files, are also included in the distribution (one of the longer-running problems is currently maintained as a standalone `.IO` file, which is also included).

The tests are grouped in several sets, which correspond to Java code-packages where they are written. Appendix A gives descriptions of all 47 short tests and the five long tests. Here, we give a brief description of the test packages:

- `advection`: Six advection tests.
- `analyticsoln`: Four tests of analytic solutions.
- `energy`: Two tests that solve the energy equation, without or with the momentum equation, with liquid water and ice, treated as fluids.
- `heattransfer`: Four heat transfer cases.
- `mpflow`: Twelve tests of various multiphase flow cases.
- `particle`: Thirteen short-running tests of solid materials, twelve of which use the MPM/PIC particle method. Also, the five longer-running problems, all of which use MPM/PIC.
- `species`: Two tests of species transport.
- `miscellaneous`: Four additional tests.

As discussed in Section 3.5, an easy way to create an input file for a new project is to use one of the "`.IO`" files created from running the tests.

3.5. Sample Project

In this section we create and run a problem that involves a solid projectile (solid 1) impacting a target (solid 2), with air in the background, on a 2D grid of rectangles. We first create our desired computational grid in the form of two "mesh files" that specify the grid nodes' coordinates and their connectivity; then we pick a suitable input file from the test suite to use as an initial template, and modify that file with CartaBlanca's Graphical User Interface (GUI) according to our desired problem specifications.

CartaBlanca has a set of Java methods that create mesh files for simple geometries, including 1D lines, 2D rectangular regions, and 3D boxes. At this time, the code does not have a GUI interface to create mesh files, or the capability to convert files from common mesh-generators to the METIS mesh format that the code uses.

The main CartaBlanca input file that specifies a problem to be run with the mesh is, by default, called `inputSpecifier.IO`. Creation of `inputSpecifier.IO` files is facilitated by use of the CartaBlanca Graphical User Interface (GUI). One can create a new input file by using the GUI to modify an existing `inputSpecifier.IO` file, or start from scratch and use the GUI to create a brand new `inputSpecifier.IO`. File `inputSpecifier.IO` includes the names and locations of the problem's mesh files.

The distribution includes directory

```
cartablanca/meshes/2D/QUADS/
```

To create mesh files for a 2-D region [0, 5] by [0, 5] ($0 \le x \le 5,\ 0 \le y \le 5$) with uniform spacing 0.5, and put them under `cartablanca/meshes/2D/QUADS/my5x5/`, first create the subdirectory `my5x5`. Then, edit the file `Create2DMesh.java` in the directory

`cartablanca/src/gov/lanl/cartablanca/main/generatemesh`, to set

```
xleng = 5.0
yleng = 5.0
numxnodes = 11
numynodes = 11
String dir = "meshes/2D/QUADS/my5x5/"
```

Compile and run `Create2DMesh.java` from the Unix (or Windows/DOS) command line, or use an IDE; the procedure is analogous to compiling and running the main code or the GUI. For example, a script to run `Create2DMesh.java` from the Unix command line could include the single line:

```
java -mx512m -classpath $CBROOT/classes
gov.lanl.cartablanca.main.generatemesh.Create2DMesh
```

After the mesh files are created, the second step is to create a suitable `inputSpecifier.IO` for the project. An easy way for this task is to modify one of the `.IO` files written to directory `testIO` by the test code. In CartaBlanca, solid-fluid interaction simulation is done using the particle-in-cell method. Thus, if one considers the package `gov.lanl.cartablanca.test.particle` (see Appendix A), it appears that the test `BulletPlateTest` is similar to the problem we wish to model. This test is a case where a bullet penetrates a plate with air in the background. Therefore, one goes to directory `~myhome/cartablanca/testIO`, and enters

```
>cp testBulletPlate.IO ../inputSpecifier.IO
```

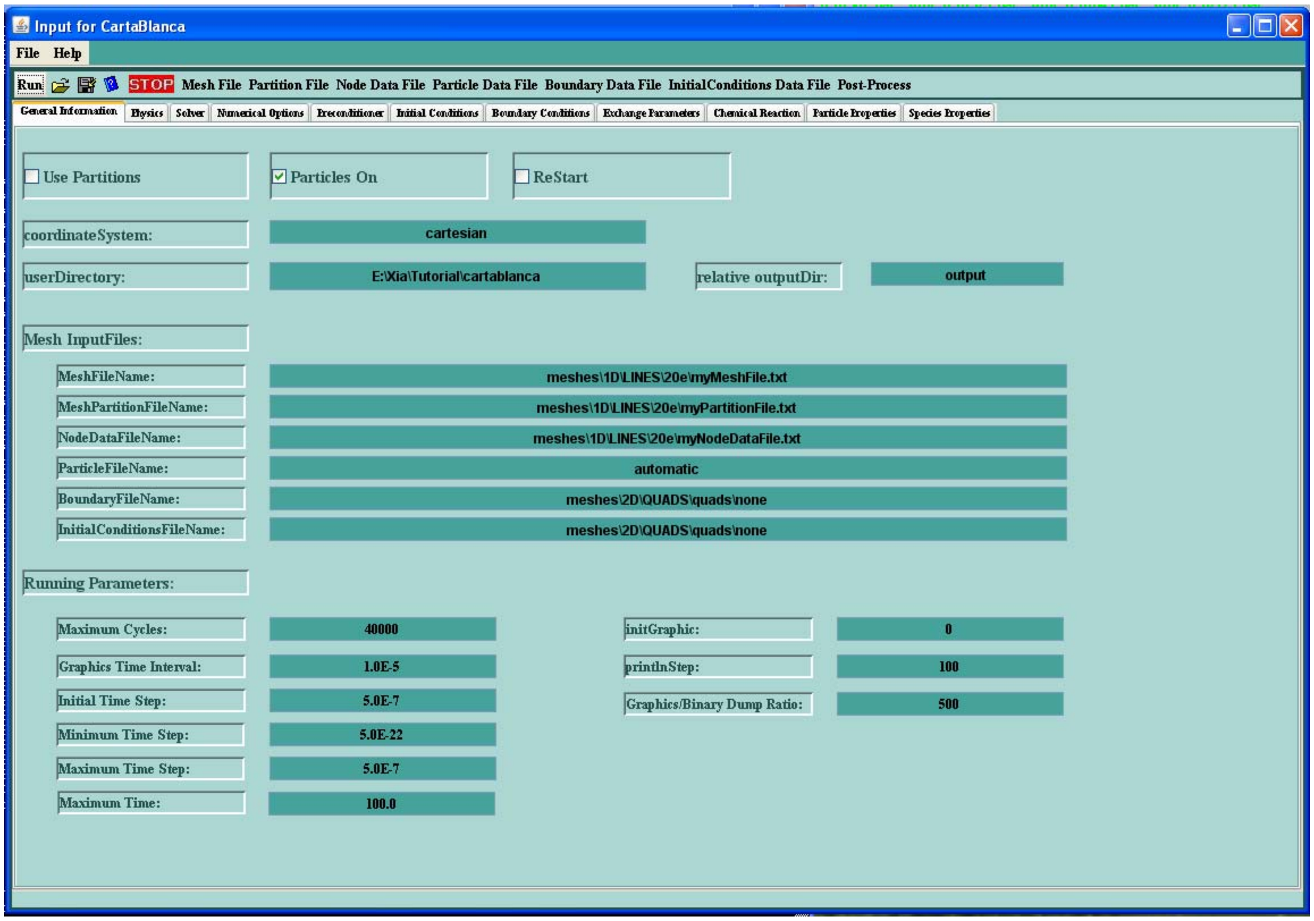
This copied file `testBulletPlate.IO` into a file `inputSpecifier.IO` in directory `cartablanca`, which can be used as the input to the GUI and the main code. This name is the default input file name in the script `runPhysMain.unix`. The default output directory is "output" under dir cartablanca/. You can overwrite it by attaching it as second arguments after inputSpecifier.

Next, one needs to modify file `inputSpecifier.IO` for the current problem. The CartaBlanca GUI is very useful for doing this. To run the GUI from the Unix command line, go to directory `~myhome/cartablanca`, and enter

```
>./scripts/unix/runRunGUI.unix
```

(Note that the environment variable `CBROOT` must be defined; see the comments in the script.)

In a Windows system, one can double-click file `rungui.cmd` in directory `cartablanca` to run the GUI, or with a software development tool (IDE), run your project with Main class `gov.lanl.cartablanca.main.RunGUI`, VM parameter `-mx512m`, and with Application parameter `inputSpecifier` (this is the configuration of the JBuilder project file `rungui.jpr`; other IDEs will have similar options). The GUI's startup display should look like Figure 2.



Use Partitions [ ] Particles On [✓] ReStart [ ]

coordinateSystem: cartesian

userDirectory: E:\Xia\Tutorial\cartablanca relative outputDir: output

Mesh InputFiles:

MeshFileName:	meshes\1D\LINES\20e\myMeshFile.txt
MeshPartitionFileName:	meshes\1D\LINES\20e\myPartitionFile.txt
NodeDataFileName:	meshes\1D\LINES\20e\myNodeDataFile.txt
ParticleFileName:	automatic
BoundaryFileName:	meshes\2D\QUADS\quads\none
InitialConditionsFileName:	meshes\2D\QUADS\quads\none

Running Parameters:

Maximum Cycles:	40000	initGraphic:	0
Graphics Time Interval:	1.0E-5	printlnStep:	100
Initial Time Step:	5.0E-7	Graphics/Binary Dump Ratio:	500
Minimum Time Step:	5.0E-22		
Maximum Time Step:	5.0E-7		
Maximum Time:	100.0		

Figure 2. Graphical User Interface startup display.

Section 4 of this document gives complete details on CartaBlanca's input specifications and the use of the GUI; here we give a basic introduction.

Before modifying the input specification file with the GUI, one can run the problem given by the current `inputSpecifier.IO` (`testBulletPlate.IO`) to get an introduction to running CartaBlanca and further test the setup and environment. From the Unix command line, under `cartablanca`, enter:

```
>./scripts/unix/runPhysMain.unix
```

In an IDE, set the Main class to `gov.lanl.cartablanca.main.PhysMain`, set the VM parameters to `-mx512m` and `-server` (the Java `-server` option improves runtime), set the Application parameter to `inputSpecifier`, and run your project (this is the configuration of the JBuilder project file `cbphysmain.jpr`; other IDEs will have similar options).

It takes only a few seconds to run this test problem.

As a calculation proceeds, CartaBlanca sends status messages to standard output. The last few lines of this output for `testBulletPlate.IO` should look like

…

…
…

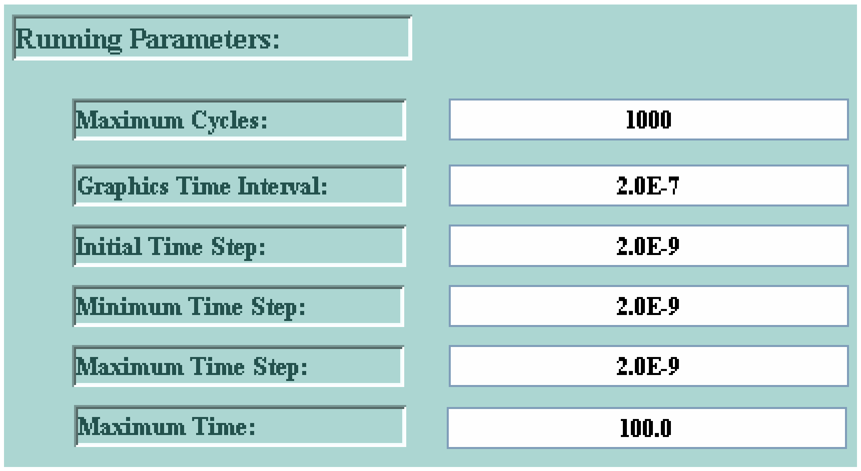
```
  n = 00020  t =  4.00000E-008  dt =  2.00000E-009,  (0)

Dumping to file E:\cartablanca\output\dump.0.00001.dfl

Just wrote E:\cartablanca\output\dump.0.00001.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase2.0.00001.dfl

Just wrote E:\cartablanca\output\dump.gridPhase2.0.00001.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase3.0.00001.dfl

Just wrote E:\cartablanca\output\dump.gridPhase3.0.00001.dfl

Done in Partition 0

Time for executing the problem: 8093 milliseconds.

Grind Time is 240 microseconds/cycle/node
```

Assuming `PhysMain` runs ok with the current input file `testBulletPlate.IO`, one now is ready to create a new input file by modifying `testBulletPlate.IO` (as `inputSpecifier.IO`) with the GUI. In all of the following, be sure to press the "Enter" key after entering data.

On the first panel of the GUI (**General Information**), enter for the input parameters

> **MeshFileName**, **MeshPartitionFileName**, **NodeDataFileName**,
>
> `meshes\2D\QUADS\my5x5\myMeshFile.txt`,
> `meshes\2D\QUADS\my5x5\myPartitionFile.txt`, and
> `meshes\2D\QUADS\my5x5\myNodeDataFile.txt`, respectively.

This will use the computational mesh created above; CartaBlanca will read the files in directories relative to its execution directory. Figure 3 shows the part of the **General Information** panel that provides overall control of a calculation. Parameters **Maximum Cycles** and **Maximum Time** specify the duration of the calculation (in problem-time seconds), according to whichever is reached first. Parameters **Initial Time Step**, **MinimumTime Step**, and **Maximum Time Step** may need to be modified based on requirements of accuracy and stability. **Graphics Time Interval** controls how often to output data files. Figure 3 shows a problem set to run from 0.0 s to 2.0 x 10^{-6} s, with 11 graphics edits. Also, note that on the first line of the first panel, the checkbox **Particles On** is checked because the bullet-plate problem uses CartaBlanca's Particle-In-Cell method to represent solids.


Running Parameters:

Maximum Cycles:	1000
Graphics Time Interval:	2.0E-7
Initial Time Step:	2.0E-9
Minimum Time Step:	2.0E-9
Maximum Time Step:	2.0E-9
Maximum Time:	100.0

Figure 3. Calculation control parameters.

The second tab of the GUI (**Physics**) brings up a panel that specifies the physical processes to be modeled and the CartaBlanca algorithms to be used for their solution (e.g., choice of flow system for momentum transport), as well as supporting data such as physical constants. Parameters **numNonParticleMaterials** and **numParticleMaterials** specify the number of phases that are to be modeled with CartaBlanca's ALE algorithm and PIC (MPM) algorithm, respectively. (A total of four phases can be modeled, each of which can contain a number of species.) The current input file uses two particle materials (an aluminum plate and a lead bullet) and one fluid material (air); it also assumes that only mechanical properties such as velocity, deformation, etc. are of interest, so only the momentum equation is solved. If temperature is to be considered, one needs to check the item **solveEnergyTransport**, and choose an energy system by selecting a suitable model from the list under **Choose energySystem** (the default is `NLEnergyBasic`). The PIC method should be used to model solid materials; in this case **flowSystem** should be `NLMultiPhaseFlowPexp` (pressure solution is explicit in time). Currently, an implicit particle method is not available. A complete description of the models available in the **Physics** panel is given in Section 4.

The sixth panel of the GUI (**Initial Conditions**) is used to specify the problem's initial geometry, material composition, and starting material properties (velocities, temperatures, etc.). **Initial Conditions** contains two subpanels: **Regions Definition** and **Regions Data**. Figure 4 shows the **Regions Definition** subpanel, which is used to break up the computational domain into sub-regions that are occupied by the individual problem components at the start of a calculation.

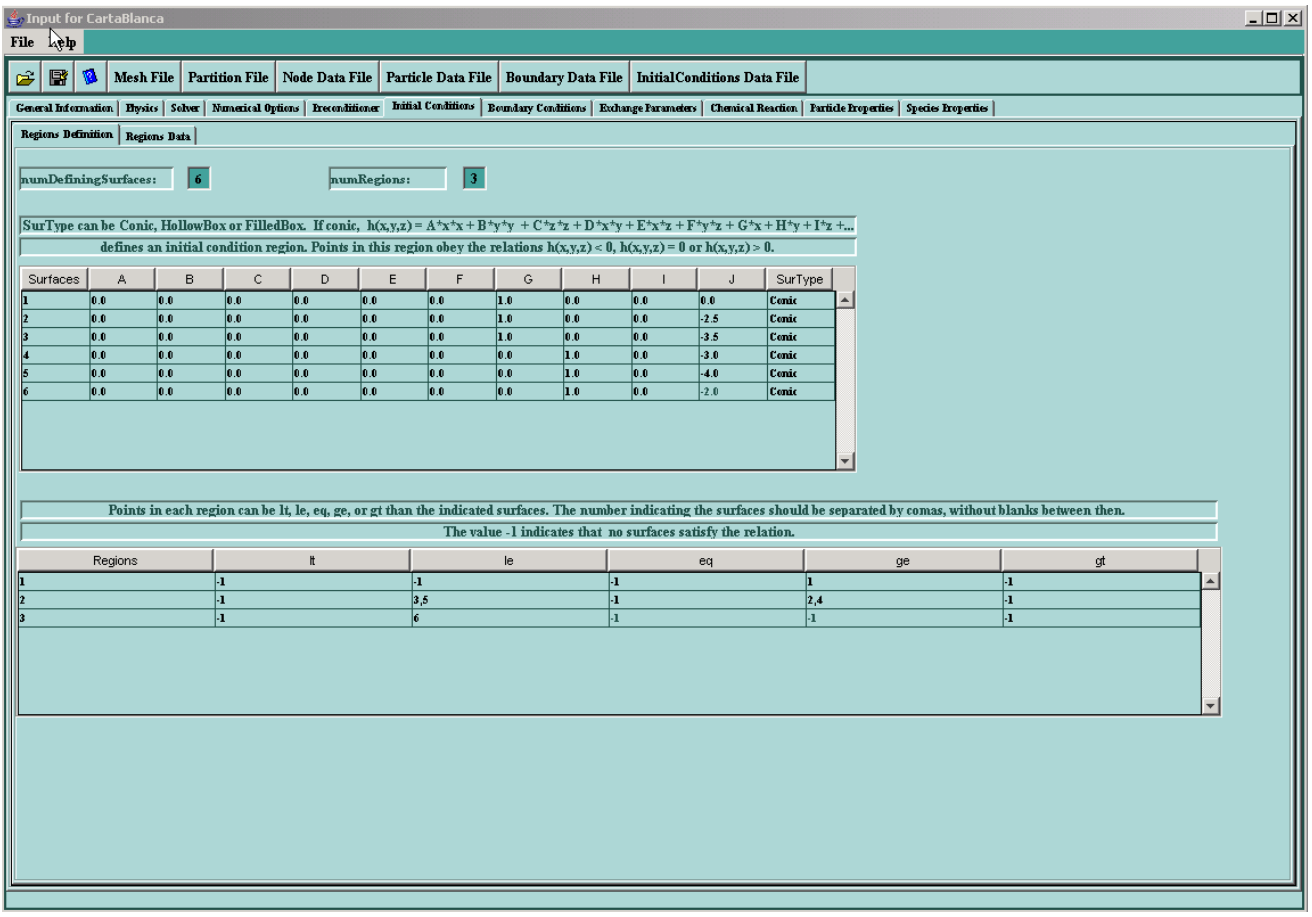
As an example, assume that the projectile originally occupies the region [2.5, 3.5] by [3, 4] ($2.5 \le x \le 3.5,\ \ 3 \le y \le 4$), and the target occupies the region [0, 5] by [0, 2]. We are going to define 3 initial regions: the entire domain (grid), the projectile, and the target. The initial regions are defined by combining surfaces in 3-D space; these basic surfaces are specified with the surface table, the top table on the sub-panel **Regions Definition**. We will use 6 surfaces to define our 3 regions; therefore, in **Regions Definition**, set **numDefiningSurfaces** to 6, and **numRegions** to 3. The surface table will have 6 rows, for Surfaces 1-6. The **SurType** (surface type) for all rows should be `Conic`, which is a simple way to define a surface. A conic surface in 3-D space is described by the following expression with coefficients *A, B, C, D, E, F, G, H, I, J*:

$$h(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J\ .$$

This expression will be used to define regions in which the points obey one or more of the relations

$h(x, y, z) < 0.0$, $h(x, y, z) \le 0.0$, $h(x,y,z) = 0.0$, $h(x, y, z) \ge 0.0$, $h(x, y, z) > 0.0$.

For example, the region [0, 5] by [0, 2] can be defined as: $y - 2 \le 0$, for all x in our domain. Therefore, a row of the surface table should have H =1, J = -2 (in our example, to define surface 6; see Figure 4), and the region definition table (the bottom table in **Regions Definition)**, defines a corresponding region (region 3 in our example) with `6` in the le column. Using this logic, the 3 initial regions are defined as shown in Figure 4.



numDefiningSurfaces: 6

numRegions: 3

SurType can be Conic, HollowBox or FilledBox. If conic, h(x,y,z) = A*x*x + B*y*y + C*z*z + D*x*y + E*x*z + F*y*z + G*x + H*y + I*z +...
defines an initial condition region. Points in this region obey the relations h(x,y,z) < 0, h(x,y,z) = 0 or h(x,y,z) > 0.

Surfaces	A	B	C	D	E	F	G	H	I	J	SurType
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	Conic
2	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	-2.5	Conic
3	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	-3.5	Conic
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-3.0	Conic
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-4.0	Conic
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	-2.0	Conic

Points in each region can be lt, le, eq, ge, or gt than the indicated surfaces. The number indicating the surfaces should be separated by comas, without blanks between then.
The value -1 indicates that no surfaces satisfy the relation.

Regions	lt	le	eq	ge	gt
1	-1	-1	-1	1	-1
2	-1	3,5	-1	2,4	-1
3	-1	6	-1	-1	-1

Figure 4. GUI "Regions Definition" sub-tab.

In the surface table, surface 1 defines $x - 0$; surface 2, x - 2.5 ; surface 3, x - 3.5 ; surface 4, $y -$ 3.0 ; surface 5, $y - 4.0$; and surface 6, $y - 2.0$. The six surfaces are used to define 3 initial regions in the lower region definition table. Region 1 is defined with the value `1` in the **ge** column, and `-1` in all other columns; this specifies that only surface 1 is used, with >= , defining a region $x - 0 \ge 0$, or the entire grid. Region 2 has values `3,5` in the **le** column, and `2,4` under **ge**; this specifies surfaces 3 and 5 with $h(x, y, z) \le 0.0$, and surfaces 2 and 4 with $h(x, y, z) \ge 0.0$, defining a region $2.5 \le x \le 3.5,\ 3 \le y \le 4$. Similarly, region 3 is $y \le 2$.

The starting regions defined above are initialized with the **Regions Data** subpanel of the **Initial Conditions** panel. This in turn contains subpanels **RD: Material 1**, **RD: Material 2**, **RD: Material 3**, **RD: Material 4**, and **RD: All Materials**. The first four tabs are used to set material (phase)-

specific initial conditions, where again we note that a CartaBlanca phase can contain more than one species. (The materials and species themselves are specified in the **Species Properties** panel of the GUI.) **RD: All Materials** is used to set initial values for the common pressure and for the turbulence K, ε model, for each of the regions. The material-specific tabs are used to set initial values for volume fraction (e.g., **vfrac1** for material 1), velocity (**U1**, **V1**, etc), temperature (e.g., **T1**), and species mass fraction (**s1MF1**, etc.) if a phase contains more than one species. The variables **DX1**, etc. (for displacements), are now only used in a special physics module. Again, these values are all set for each individual initial region.

Initialization is done sequentially, in the order region 1, …, region N. Thus, although in our example region 1 includes regions 2 and 3, any initialization done to regions 2 and 3 during the initialization of region 1 will be overwritten when initializing regions 2 and 3. In other words, the initialization done to region 1 has effect only in the part of region 1 that does not overlap region 2 or 3.

The new problem is ready for a trial run. The third button in the GUI's toolbar (Figure 5) must be pressed to update the current `inputSpecifier.IO` (the original will be overwritten). Figure 6 (in Section 3.6) shows the starting configuration of the problem.

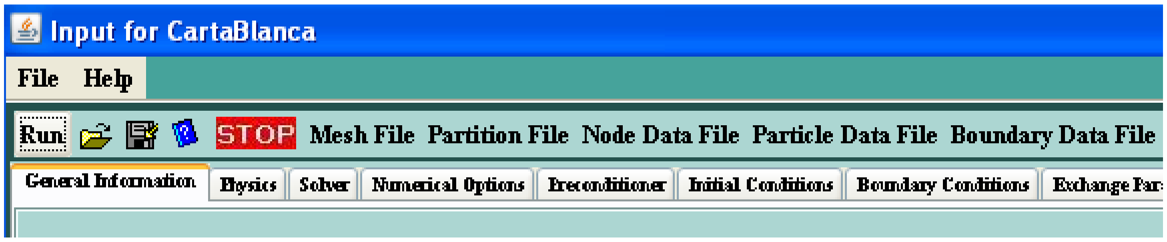


Figure 5. Saving the new input file.

In the remainder of this section we give an overview of the **Boundary Conditions** panel, which is closely related to **Initial Conditions**, and show an alternate way to input initial and boundary data.

Problem boundary conditions are specified with the GUI **Boundary Conditions** panel, using surfaces and regions that are defined in the **BcDefinitions** subpanel, in a manner similar to the initial conditions surfaces and regions. In addition, the **type** and **kind** of the boundary condition regions are specified, where **type** can be `internal` or `external`, and **kind** can be `wall`, `reflective`, `reflcorner`, `inflow`, `outflow`, `inflow-outflow`, `pressure`, or `vel-direction`. The boundary conditions are specified with the **BcData** subpanel, which has tabs **AllFluids**, **Material 1**, **Material 2**, **Material 3**, and **Material 4**.

If no boundary region is set, all the geometric boundaries in the problem geometry are considered to be default `wall` boundaries. For `wall` boundaries, by default the outward normal velocity is set to zero and inward normal velocity and tangential velocity are allowed. For the energy module, a `wall` boundary is adiabatic unless otherwise specified by the user. The `wall` boundary condition for temperature T is assumed to have the form

$$k\frac{\partial T}{\partial n} = -h(T - T_\infty) + q$$

where k is the heat conductivity as given in the energy equation, n is the outward normal direction, h is the heat transfer coefficient, T_∞ is the ambient temperature, and q is the heat flux. On **BcData** subpanel **Material 1**, Table I has columns labeled **TempH**, **TempPhi**, and **TempFl**, which correspond to h, T_∞, and *q* respectively in the temperature boundary condition. Suppose one only wants to solve the energy transport equation and wants to set the boundary temperatures to T = 300 and T = 500, say, in the two regions $y - 0 = 0$ and $y - 1 = 0$ for material 1 and material 2, respectively (assuming only two materials are used): enter a large number such as `1.0E20` in the first and second rows under **TempH**, and enter `300.0` and `500.0` under **TempPhi**. This effectively sets the boundary temperatures to the desired values. A description of CartaBlanca boundary condition usage is given in Section 4.7.

If an initial region or boundary region has a complicated geometry, one can also optionally create an initial or boundary data file to set the region. An example is in file `Poiseulle1_RF.IO`, which is written by test-suite problem `testPoiseuille1_RF_Test`. Its GUI **Boundary Conditions** panel has, for region 2, `-1` entered for all columns **lt**, **le**, etc. This triggers the use of a boundary data file, which is specified in the **General Information** panel, where **BoundaryFileName** is

```
 meshes\2D\QUADS\Poiseuille\myBCFile.txt
```

This file contains

```
 1
 1 wall 2
 10.0 0.0 0.0
 10.0 1.0 0.0
```

The first line gives the number of boundary sections in the file, in this case, `1`. In the second line, the first number, `1`, indicates the boundary section index; these indexes start at 0, thus, this boundary section is the second boundary section. The name `wall` is the boundary kind, and `2` gives the number of nodes in this boundary section. The following two lines give the coordinates of the nodes (*x, y, z)*. The initial condition data files have a similar format, but without the boundary-kind specification.

3.6. Calculation Results

As a calculation proceeds, CartaBlanca writes graphics-output files in Tecplot format in directory `cartablanca/output`, according to the edit interval specified by the **Graphics Time Interval** field in the **General Information** panel. The sample problem was originally set to write 11 edits over the time interval 0.0 s – 2.0 x 10^{-6} s. After (or while) running the problem, opening the files

```
cartablanca/output/gridPhase2partition0-00000.dat
```

and

```
cartablanca/output/gridPhase3partition0-00000.dat
```

in Tecplot brings up time = 0.0 s plots of the projectile and target, respectively, as shown in Figure 6.

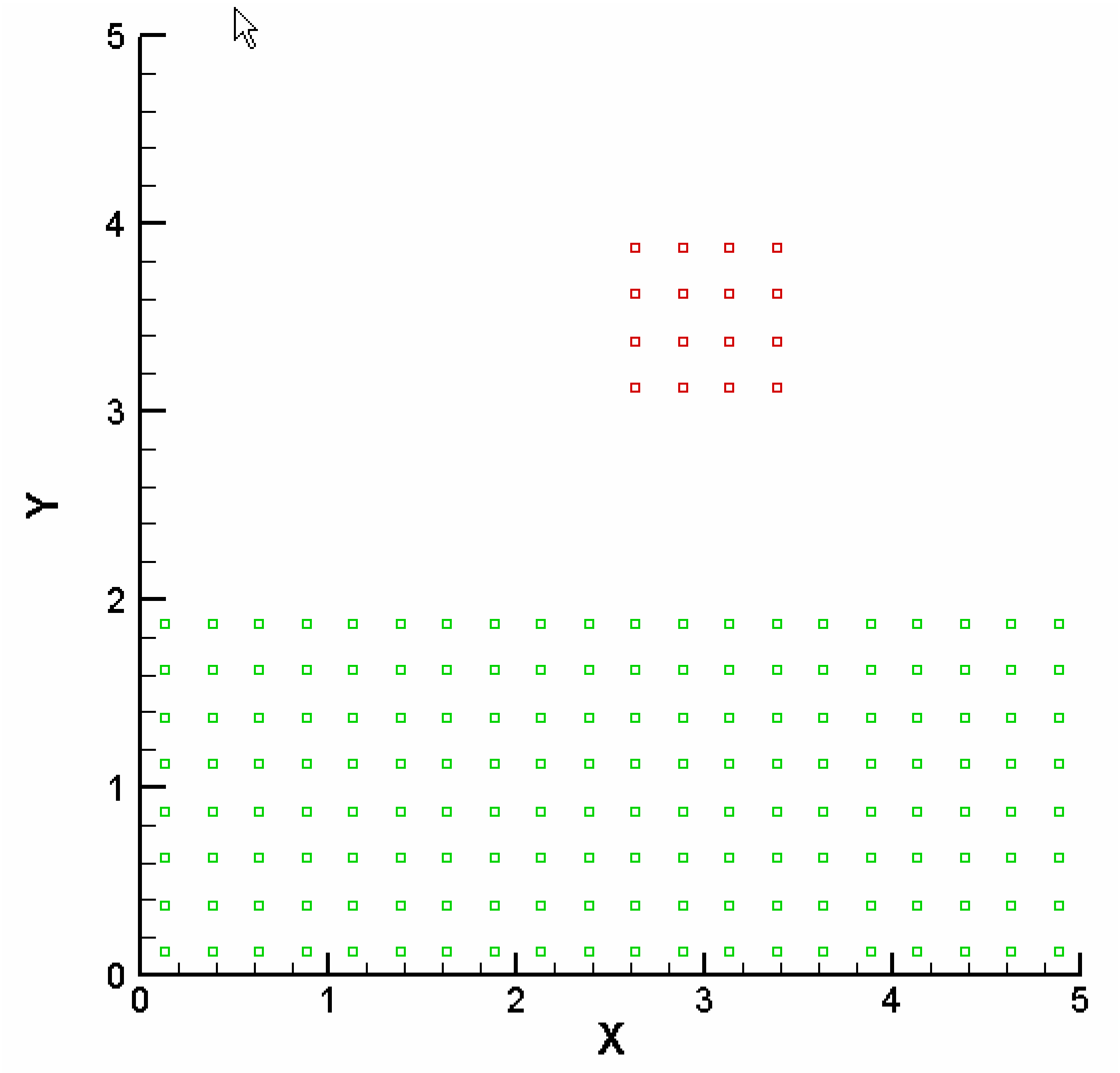


Figure 6. Sample problem projectile and target.

Here we have used Tecplot's scatter mode, to show the actual calculational particles CartaBlanca used for its particle-in-cell representation of the projectile and target. Figure 5 only shows their initial locations, which were specified according to the discussion above on initial conditions. (The actual number of particles is determined by the specified mesh and the GUI **Particle Properties** panel.) Many other parameters are written out to the graphics files.

A complete description of CartaBlanca's output is given in Section 6.

4. INPUT PREPARATION AND SPECIFICATIONS

The main input that specifies and controls a CartaBlanca calculation is contained in a text file called, by default, `inputSpecifier.IO`. File `inputSpecifier.IO` also contains the names of six additional ("Mesh Input") files, two of which the user must provide to specify the computational nodalization, one of which is required for parallel runs to provide the mesh partitioning, and three of which are optional files that give particle, initial condition, and boundary condition data. While an `inputSpecifier.IO` file can always be edited by hand (and we encourage users to have a look at one), the CartaBlanca GUI makes development of an input file vastly easier and less error-prone. The GUI can read and modify an existing `inputSpecifier.IO`, and it also can create one from scratch starting with a set of default parameters.

The GUI is organized in a hierarchy of standard, familiar tools (buttons, tabs, and menus). Figure 7 shows the highest level, which consists of three rows:

- "`File`" and "`Help`" provide standard menus (currently only "`File`" → "`Exit`" is available).

- Twelve buttons are on the second row (the toolbar). The first (leftmost) button is a new feature in the GUI: it runs the `inputSpecifier.IO` in the user directory (`cartablanca`, by default). The second button is currently not operational: it would open a desired file. The third must be pressed (clicked) by the user to save the current `inputSpecifier.IO` to the user directory. The fourth button also brings up "help" (currently not implemented). The fifth button (also new) stops the run. The following six buttons bring up file browsers for selection of the Mesh Input files (which can, alternatively, be specified elsewhere in the GUI; see Section 4.1). The last button on the toolbar, "Post-Process" (also new), brings up an explorer in the running directory, so that post-processing macros can be activated conveniently. The format and contents of the Mesh Input files are described in Section 4.4.1.

- The third row is the main entry into the GUI; it contains tabs (currently 11) that display panels for input of the problem's physics and control data. These panels are organized to contain data related to the various aspects of CartaBlanca's logic and capabilities, and most of them contain sub-panels.

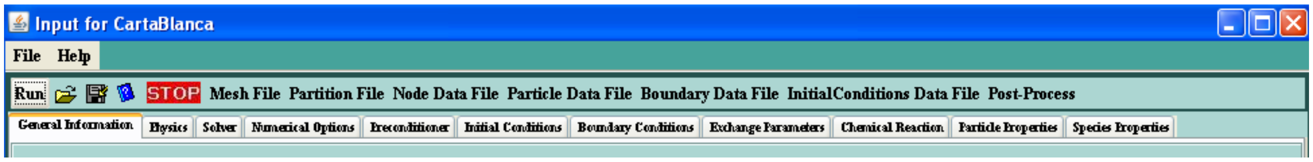


Figure 7. GUI's highest control level.

The 11 highest-level tabs (and corresponding panels) are:

- General Information – specifications of global problem data (e.g., use of the particle-in-cell method, names of the Mesh Input files, time step size, edit frequency).

- Physics – the physical processes to be modeled and the solution algorithms (e.g., choice of flow system); physical constants.

- Solver – selection of equation solver(s) and related parameters.

- Numerical Options – switches for specific options for the ALE and PIC/MPM numerics (e.g., artificial viscosity); advection Courant number.

- Preconditioner – selection of quantities to precondition to reduce the number of Krylov iterations; setting the preconditioner algorithms and related parameters.

- Initial Conditions – specification of regions in the problem domain and their initial (time = 0) setup (e.g., materials and their velocities, pressures, etc.).

- Boundary Conditions – specification of regions and boundary conditions to be applied.

- Exchange Parameters – momentum, energy, and mass exchange data, according to the number materials (fields) in the problem.

- Chemical Reaction – data for any reactions to be modeled (e.g., Arrhenius activation energy, specification of reaction and product phases).

- Particle Properties – number of particles per cell (for PIC/MPM calculations), damage-calculation switch.

- Species Properties – selection from built-in material constitutive models (e.g., Kelvin, Johnson-Cook), and assignment of constitutive-model data.

The input specifications that follow, in Sections 4.1 – 4.11, are organized according to the CartaBlanca GUI tabs. Section 4.1.1 describes the Mesh Input Files.

The GUI data are comprised of text fields (either keywords or user-supplied, such as file names), reals (floating point), integers, and booleans (typically entered with checkboxes). Keyword entry is in most instances facilitated by selection from built-in dropdown lists. Where an input parameter requires a real value, exponential notation may optionally be used (e.g., 1.678E12).

If the GUI is started without an `inputSpecifier.IO` file, the GUI input parameters will be initially set to default values that are specified in the CartaBlanca coding.

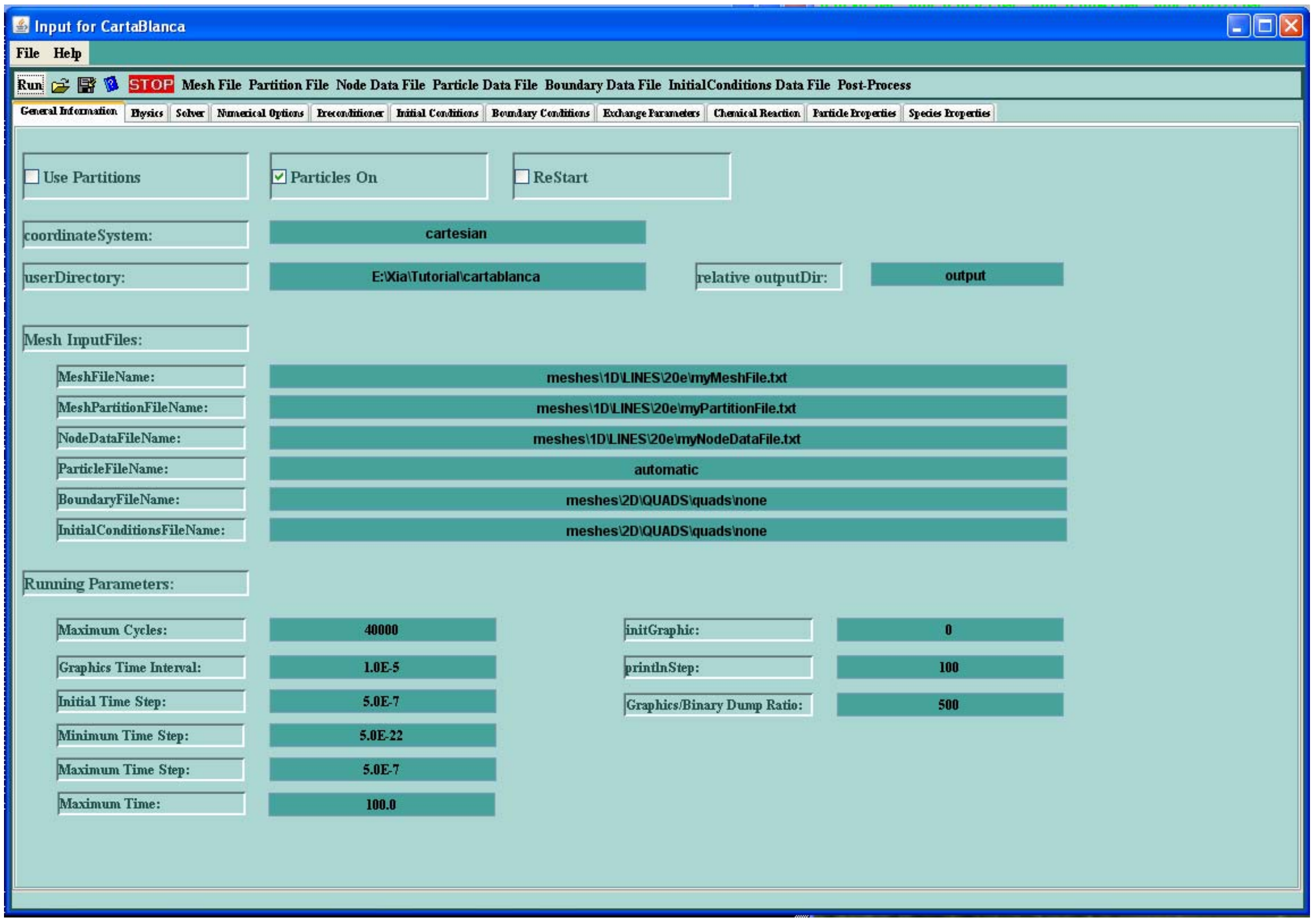
4.0.1. Systems of Units

CartaBlanca makes no assumptions as to a units system; there is no switch in the input file for units selection. The only user requirement is that all input for a model adhere to a self-consistent system. Most input models developed at Los Alamos have been in the cgs system.

4.1. General Information

The **General Information** tab brings up a panel that is used to specify the global controlling parameters for a CartaBlanca calculation, such as mesh files, calculation length, time-step size, and data output interval. Other data entered on the **General Information** panel include choice of a parallel (multiprocessor) calculation with mesh partitions, use of the MPM/PIC particle method, and the coordinate system. Also, a restart from a previous run can be indicated, and the user directory and an output directory relative to the user directory can be specified. Figure 8 shows a typical General Information panel.

Following are specifications for the **General Information** input data.


Input for CartaBlanca
File Help
Run STOP Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File Post-Process
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties

Use Partitions [ ] Particles On [✓] ReStart [ ]

coordinateSystem:	cartesian
userDirectory:	E:\Xia\Tutorial\cartablanca
relative outputDir:	output

Mesh InputFiles:

MeshFileName:	meshes\1D\LINES\20e\myMeshFile.txt
MeshPartitionFileName:	meshes\1D\LINES\20e\myPartitionFile.txt
NodeDataFileName:	meshes\1D\LINES\20e\myNodeDataFile.txt
ParticleFileName:	automatic
BoundaryFileName:	meshes\2D\QUADS\quads\none
InitialConditionsFileName:	meshes\2D\QUADS\quads\none

Running Parameters:

Maximum Cycles:	40000	initGraphic:	0
Graphics Time Interval:	1.0E-5	printlnStep:	100
Initial Time Step:	5.0E-7	Graphics/Binary Dump Ratio:	500
Minimum Time Step:	5.0E-22		
Maximum Time Step:	5.0E-7		
Maximum Time:	100.0		

Figure 8. GUI "General Information" tab.

<u>**Use Partitions**</u>: boolean; if checked, the input will specify a partitioned mesh, and a parallel calculation will be run. Otherwise, a serial calculation will be run. Running CartaBlanca in parallel mode is described in Section 5.3.

<u>**Particles On**</u>: boolean; if checked, the run will utilize the PIC (MPM) algorithm for solution of at least one material (phase). Otherwise, the ALE method will be used for all materials (phases).

ReStart: boolean; if checked, the run will be a restart from the results (dump file or files) of a previous run. The initialization of a restart run from dump files is described in Section 5.2 (see also input variable **initGraphic**, below in this section).

coordinateSystem: keyword text; the coordinate system to be used, either `cartesian`, `cylindrical`, or `spherical`. By default, in a 2D cylindrical coordinate system, the *y*-axis is the axis of rotational symmetry.

userDirectory: text; the absolute path of the CartaBlanca directory where the problem will be run. Automatically set to the current GUI directory.

relative outputDir: text; the output directory path relative to **userDirectory**. The default is `output`. This can be overwritten in a run script by adding a new output directory as the second argument (after the input file; command line arguments for running the code are described in Section 5.). This is a new feature in the GUI.

Mesh Input Files: Six text fields that give the directory locations and names of files that specify the computational domain and related data. Directories may be relative to **userDirectory**. Alternately, one or more of these files can be chosen by using the six correspondingly-named buttons in the main toolbar at the top (second line) of the GUI. Specifications for the Mesh Input Files are given in Section 4.1.1. Note that, while use of some of these files is optional, none of the six fields here should be entirely blank.

> **MeshFileName**: file that defines the computational mesh elements (e.g., 2-D quadrilaterals, 3-D hexahedra), by their individual vertex nodes.
>
> **MeshPartitionFileName**: file that assigns each of the mesh elements to one of two or more partitions of the domain, which are assigned to parallel processors; only needs to be specified for a parallel calculation. A discussion of CartaBlanca's parallel processing capabilities is given in Section 5.2.
>
> **NodeDataFileName**: file that provides the coordinates of the mesh-element vertex nodes.
>
> **ParticleFileName**: file that provides initialization data for computational particles; only required for calculations that use the PIC (MPM) method. Alternately, the keyword text `automatic` can be entered for default settings. Additional particle input is described in Section 4.10.
>
> **BoundaryFileName**: file that specifies boundary condition locations. Section 4.7 gives details on boundary condition specifications and usage, including an alternate way to provide the boundary condition location information, using the GUI.
>
> **InitialConditionsFileName**: file that specifies initial condition locations. Section 4.6 describes an alternate way to specify this information, using the GUI. Section 4.6 gives details on CartaBlanca's initial condition setup.

Running Parameters: Nine fields, four integer and five real (floating point), that specify overall calculation behavior.

Maximum Cycles: integer; the maximum number of time steps for this run; calculation will terminate when this is exceeded, or Running Parameter **Maximum Time** is exceeded (see below) - whichever is satisfied first.

Graphics Time Interval: real; the time interval between writing of graphics edit files. Section 6.2 describes CartaBlanca's graphics-edit files. Also, in conjunction with Running Parameter, **Graphics/Binary Dump Ratio** (see below), specifies interval between writing of restart dumps.

Initial Time Step: real; the time step size to try for the calculation's first cycle (time step).

Minimum Time Step: real; the minimum time step size allowed; the calculation will be aborted if the time step size falls below this value.

Maximum Time Step: real; the maximum time step size allowed.

Maximum Time: real; calculation will be stopped at this time, or when Running Parameter **Maximum Cycles** is exceeded - whichever is satisfied first.

initGraphic: integer; used for restart calculations to specify the dump file(s) used to initialize the calculation, and the running sequence number for the first graphics and dump edits. The initialization of a restart run from dump files is described in Section 5.2.

printlnStep: integer; the time step interval for status edits to the standard output (the screen, or as redirected to a file). Section 6.1 describes these edits.

Graphics/Binary Dump Ratio: integer; the time interval for binary dumps as a multiplier on the graphics-edit time interval (Running Parameter **Graphics Time Interval**).

Section 4.1.1 gives descriptions of the Mesh Input Files that are specified in the **General Information** panel. Section 4.1.2 describes standalone files in the CartaBlanca release package that can be used to generate node, mesh, and partition files for various geometries. Section 4.1.3 describes a code option to apply a periodic boundary condition to a mesh.

4.1.1. Mesh Input Files

In addition to file `inputSpecifier.IO`, CartaBlanca reads three required input files that specify the computational node locations, mesh (node) connectivity, and node-edge mesh partitions for parallel computation (required only for a parallel run), and optionally three files that specify boundary conditions, initial conditions, and the distribution and properties of computational particles for calculations that use the PIC/MPM logic. These files are called collectively a problem's Mesh Input Files. The six Mesh Input Files are:

`NodeDataFile`, node coordinates (required),
`MeshFile`, the mesh connectivity (required),
`MeshPartitionFile`, mesh partitioning (required for parallel runs),
`ParticleFile`, particle-model data (optional, an automatic calculation can be chosen),
`BoundaryFile`, boundary condition nodes and types (optional, can be given in `InputSpecifier.IO`), and

`InitialConditionsFile`, initial condition nodes (optional, can be given in `InputSpecifier.IO`)

These file names are not required; each of the Mesh Input files can be named according to the user's wishes, as described in Section 4.1. All are text files; their specifications are as follows:

`NodeDataFile`, `MeshFile`, and `MeshPartitionFile`

These three files define the geometry of CartaBlanca's computational grid. Their formats follow from those required by the METIS mesh-partitioning program [6]. The three files contain the mesh connectivity, the node coordinates and the partitioning of the mesh elements. Please see the METIS manual [6] for additional description of these files.

The node coordinates file (e.g., `NodeDataFile)` has the format (in this case coordinates are given for a 3-D calculation):

```
1    5.000000e+00  0.000000e+00  5.000000e+00
2    5.000000e+00  5.000000e+00  5.000000e+00
3    5.000000e+00  1.000000e+00  5.000000e+00
4    5.000000e+00  2.000000e+00  5.000000e+00
5    5.000000e+00  3.000000e+00  5.000000e+00
6    5.000000e+00  4.000000e+00  5.000000e+00
7    5.000000e+00  5.000000e+00  0.000000e+00
8    5.000000e+00  5.000000e+00  4.000000e+00
9    5.000000e+00  5.000000e+00  3.000000e+00
10   5.000000e+00  5.000000e+00  2.000000e+00
11   5.000000e+00  5.000000e+00  1.000000e+00
 .   .  .  .   .   .   .   .
 .   .  .  .   .   .   .   .
 .   .  .  .   .   .   .   .
```

The real numbers in the file have a free format. A 2-D mesh has the same format as the 3D with the last coordinate equal to zero.

The connectivity file `MeshFile` represents a mesh with n elements and has n+1 lines. The first line contains information about the size and the type of the mesh. The remaining lines contain the nodes that compose each element. The information in the first line consists of two integers: the first is the number of elements in the mesh, and the second denotes the type of elements in the mesh: 1 for triangles, 2 for tetrahedra, 3 for hexahedra and 4 for quadrilaterals. The number of nodes in each of the following lines depends on the kind of element with three for triangles, four for tetrahedra and quadrilaterals, and eight for hexahedra. As an example for hexahedra:

```
125 3
72 76 117 104 77 96 153 136
76 75 113 117 96 92 154 153
75 74 109 113 92 88 155 154
74 73 105 109 88 84 156 155
73 67 97 105 84 71 140 156
104 117 118 103 136 153 157 135
117 113 114 118 153 154 158 157
113 109 110 114 154 155 159 158
```

```
    109 105 106 110 155 156 160 159
    105 97 98 106 156 140 139 160
    103 118 119 102 135 157 161 134
    118 114 115 119 157 158 162 161
    114 110 111 115 158 159 163 162
    110 106 107 111 159 160 164 163
     .   .   .   .   .   .   .   .
     .   .   .   .   .   .   .   .
     .   .   .   .   .   .   .   .
```

In the case of triangles and tetrahedra, the ordering of the nodes for each element is irrelevant. This is not the case for quadrilaterals and hexahedra for which the nodes must obey a specific order, as shown in Figure 9:

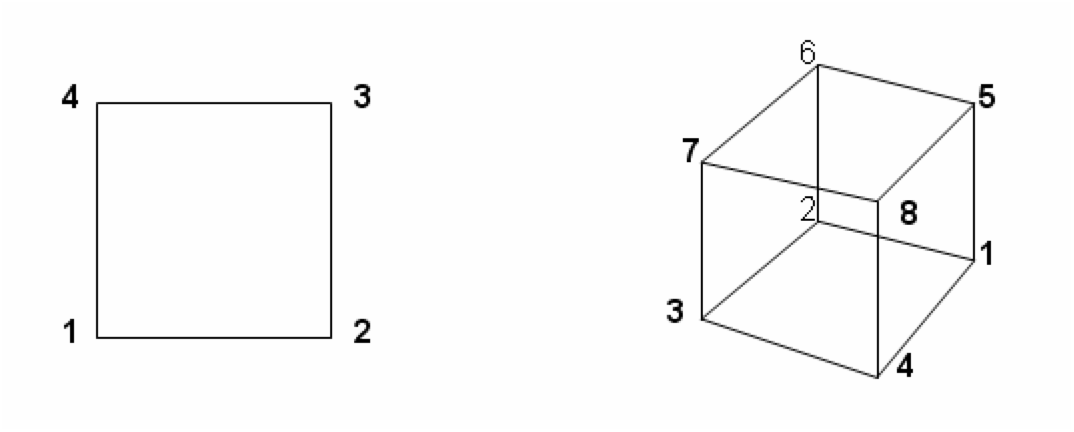


Figure 9. Node ordering for quadrilaterals and hexahedra.

CartaBlanca requires mesh partitioning to be done in such a way that elements (i.e., triangles, etc.) and not nodes are partitioned. Referring to Figure 10, the mesh partitioning for CartaBlanca must be done along node-edge connections. In the Figure, the heavier edge connections denote the boundary between partition A and partition B. To implement this mode of partitioning in CartaBlanca, nodes on the partition boundaries are duplicated. In the example in the Figure, the three nodes along the partition boundary would be present in each partition as duplicates.

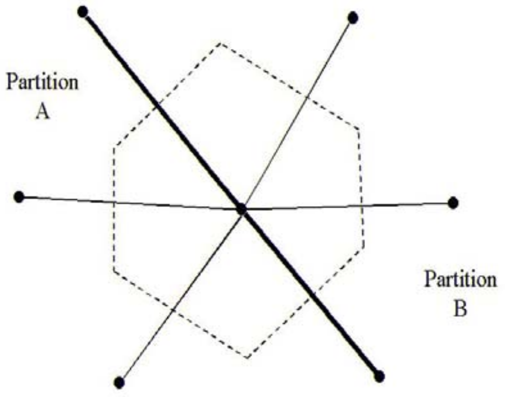


Figure 10. Partitioning in CartaBlanca; meshes must be partitioned along node connections.

The partition file has n lines for a mesh with n elements; each line has an integer representing the partition in which the element resides. The partition integers start at 0. Usually, these numbers are obtained using Metis (see also Section 4.1.2)..

To illustrate further how mesh partitioning works in CartaBlanca, a two-dimensional mesh is shown in Figure 11.

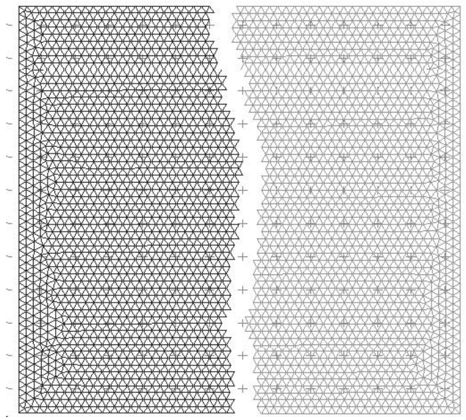


Figure 11. Two-dimensional partitioned mesh.

The mesh partitioning shown in Figure 11 was performed using the Metis program and the Metis output was then fed to CartaBlanca for computations. The actual plot was generated using the Tecplot program which operates on graphics output files from CartaBlanca (Section 6.2 gives a description of the graphics output). A further example mesh is shown in Figure 12 for the case of a three-dimensional tetrahedral mesh.

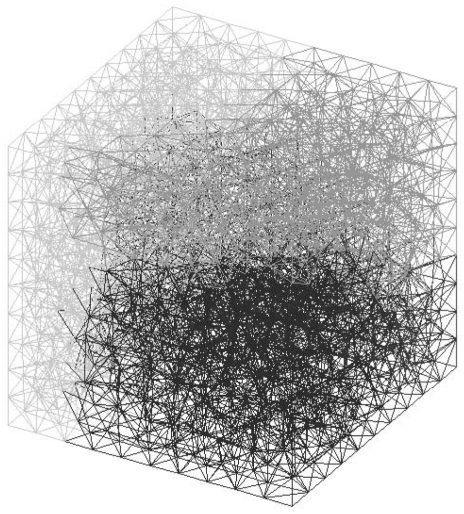


Figure 12. Three-dimensional tetrahedral element mesh. The shading denotes the 4 partitions that were computed by Metis.

Additional examples of CartaBlanca mesh partitions are shown in Figure 13.

Sections 5.2 and 7.4 have additional material on CartaBlanca parallel computing.

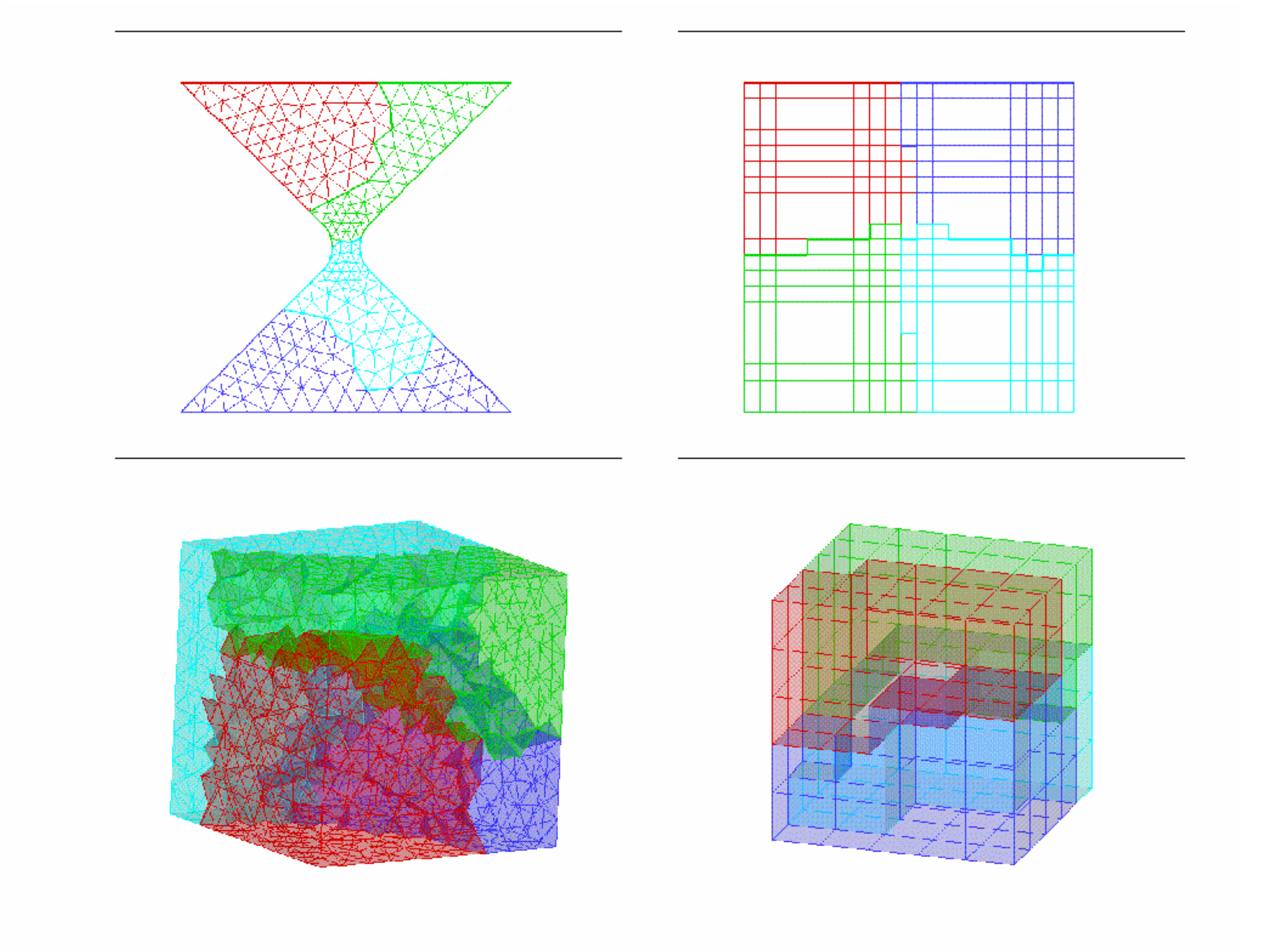


Figure 13. Additional mesh-partition examples.

The CartaBlanca distribution contains a large number of sample mesh files, in directories under the directory `cartablanca/meshes`. Often, these files are called `myNodeDataFile`, `myMeshFile`, and `myPartitionFile`, and their actual contents are indicated by the names of the directories that contain them. For example, directory

```
cartablanca/meshes/2D/QUADS/201nx144n
```

contains three mesh files that specify a two-dimensional grid of quadrilaterals, with 201 nodes for the x-coordinate and 144 nodes for the y-coordinate. Section 3.5 of this Manual shows the use of CartaBlanca itself to generate relatively simple node and mesh files. The generation of node, mesh, and partition files is further discussed below in Section 4.1.2.

<u>`ParticleFile`</u> (optional)

CartaBlanca uses the Material Point Method (MPM), an advanced version of the PIC method, for solid mechanics modeling. There are two ways to initialize an MPM calculation for a material (phase). One can provide a `ParticleFile`, and give its name and location in the **General Information** panel's **ParticleFileName** field (or browse to it using the Particle Data File button at the top of the GUI). Or, one can use the code's defaults, entering "`automatic`" in the

ParticleFileName field, and specifying the number of computational particles per mesh cell in the **Particle Properties** panel (see Section 4.10). The distribution package contains a sample `ParticleFile`

`cartablanca/particles/2d/waves.txt` ,

a snippet of which is

```
grid phase number: 1
Number of particicles: 867
Particle coordinates
  3.75100E-001  3.00100E-001
  3.75100E-001  3.12100E-001
  3.75100E-001  3.24100E-001
  3.75100E-001  3.36100E-001
  3.75100E-001  3.48100E-001
  3.75100E-001  3.60100E-001
  3.75100E-001  3.72100E-001
  3.75100E-001  3.84100E-001
  3.75100E-001  3.96100E-001
.
.
.
  6.25100E-001  8.64100E-001
  6.25100E-001  8.76100E-001
  6.25100E-001  8.88100E-001
  6.25100E-001  9.00100E-001
number of state varibales per particle: 13
particle state variables
Mass Volume U V Pressure StressXx StressXy StressYx StressYy DisplacementGradientXx
DisplacementGradientXy DisplacementGradientYx DisplacementGradientYy
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
.
.
.
  1.87500E-004  1.87500E-004  0.00000E+000 -1.00000E-002  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000 -1.00000E-002  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000 -1.00000E-002  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
  1.87500E-004  1.87500E-004  0.00000E+000 -1.00000E-002  0.00000E+000  0.00000E+000  0.00000E+000
0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000  0.00000E+000
```

`BoundaryFile` and `InitialConditionsFile` (both optional)

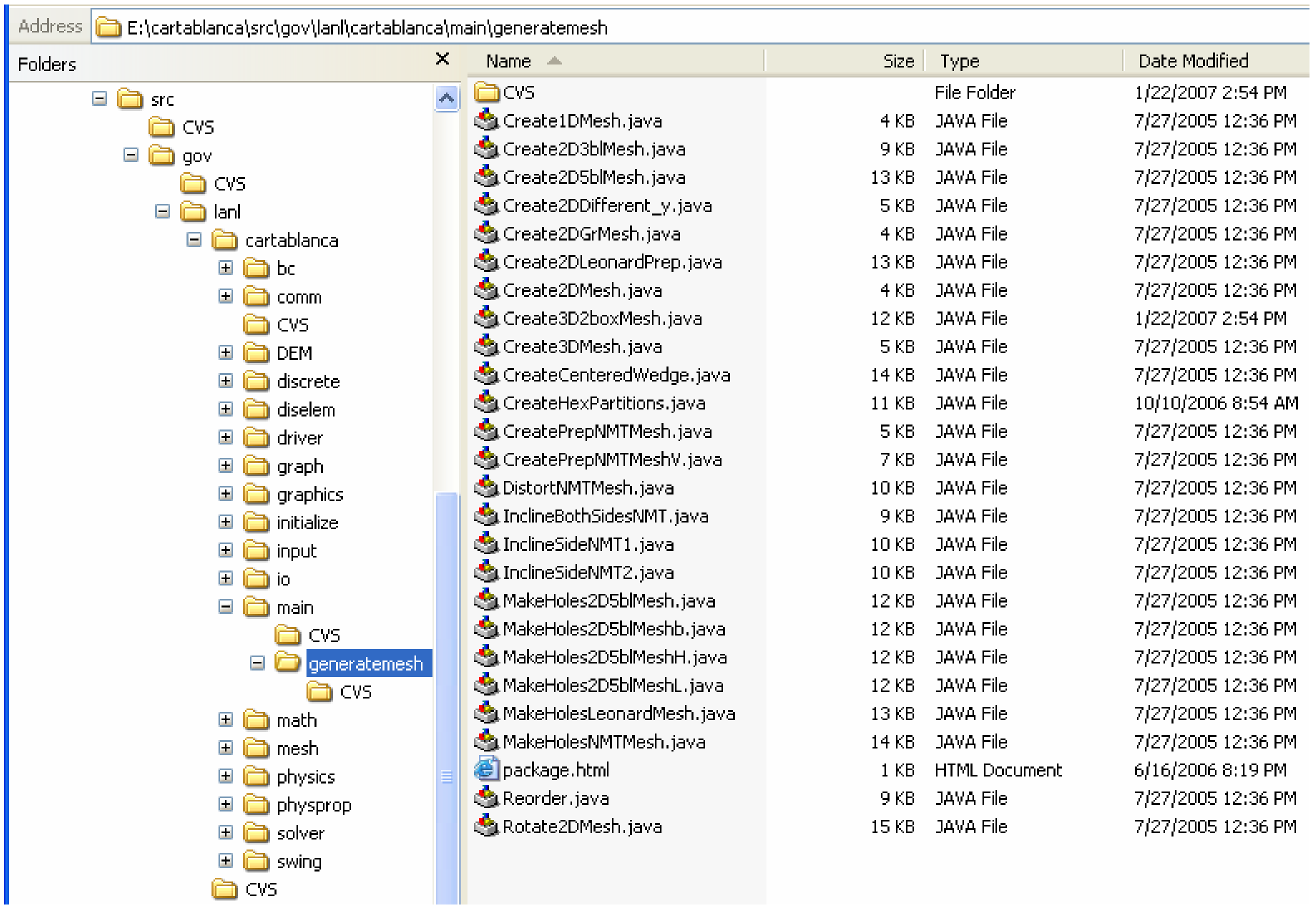
The specification of regions in the computational domain for applying initial and boundary conditions is described in Sections 4.6 (initial conditions) and 4.7 (boundary conditions). The most convenient method for such specification is the use of the GUI to set up geometries that are built-in to CartaBlanca (including conics). For initial condition regions that have shapes not suitable to this method, the user has the option to supply a file that contains node coordinates. Also, the user may wish to supply a file with boundary condition parameters (node coordinates and types), instead of using the GUI. The formats of these files are also given in Sections 4.6 and 4.7.

4.1.2. Generation of `NodeDataFile`, `MeshFile`, and `MeshPartitionFile`

The CartaBlanca release package includes a number of Java source files that can be modified, compiled, and run to generate the three files that specify the nodalization and partitioning of a problem. They are limited in the geometries that are handled, but can be quite useful nevertheless. These Java files are contained in directory

```
    src/gov/lanl/cartablanca/main/generatemesh
```

Figure 14 shows the contents of `generatemesh`.



Address E:\cartablanca\src\gov\lanl\cartablanca\main\generatemesh

Folders

```
src
  CVS
  gov
    CVS
    lanl
      cartablanca
        bc
        comm
        CVS
        DEM
        discrete
        diselem
        driver
        graph
        graphics
        initialize
        input
        io
        main
          CVS
          generatemesh
            CVS
        math
        mesh
        physics
        physprop
        solver
        swing
      CVS
```

Name	Size	Type	Date Modified
CVS		File Folder	1/22/2007 2:54 PM
Create1DMesh.java	4 KB	JAVA File	7/27/2005 12:36 PM
Create2D3blMesh.java	9 KB	JAVA File	7/27/2005 12:36 PM
Create2D5blMesh.java	13 KB	JAVA File	7/27/2005 12:36 PM
Create2DDifferent_y.java	5 KB	JAVA File	7/27/2005 12:36 PM
Create2DGrMesh.java	4 KB	JAVA File	7/27/2005 12:36 PM
Create2DLeonardPrep.java	13 KB	JAVA File	7/27/2005 12:36 PM
Create2DMesh.java	4 KB	JAVA File	7/27/2005 12:36 PM
Create3D2boxMesh.java	12 KB	JAVA File	1/22/2007 2:54 PM
Create3DMesh.java	5 KB	JAVA File	7/27/2005 12:36 PM
CreateCenteredWedge.java	14 KB	JAVA File	7/27/2005 12:36 PM
CreateHexPartitions.java	11 KB	JAVA File	10/10/2006 8:54 AM
CreatePrepNMTMesh.java	5 KB	JAVA File	7/27/2005 12:36 PM
CreatePrepNMTMeshV.java	7 KB	JAVA File	7/27/2005 12:36 PM
DistortNMTMesh.java	10 KB	JAVA File	7/27/2005 12:36 PM
InclineBothSidesNMT.java	9 KB	JAVA File	7/27/2005 12:36 PM
InclineSideNMT1.java	10 KB	JAVA File	7/27/2005 12:36 PM
InclineSideNMT2.java	10 KB	JAVA File	7/27/2005 12:36 PM
MakeHoles2D5blMesh.java	12 KB	JAVA File	7/27/2005 12:36 PM
MakeHoles2D5blMeshb.java	12 KB	JAVA File	7/27/2005 12:36 PM
MakeHoles2D5blMeshH.java	12 KB	JAVA File	7/27/2005 12:36 PM
MakeHoles2D5blMeshL.java	12 KB	JAVA File	7/27/2005 12:36 PM
MakeHolesLeonardMesh.java	13 KB	JAVA File	7/27/2005 12:36 PM
MakeHolesNMTMesh.java	14 KB	JAVA File	7/27/2005 12:36 PM
package.html	1 KB	HTML Document	6/16/2006 8:19 PM
Reorder.java	9 KB	JAVA File	7/27/2005 12:36 PM
Rotate2DMesh.java	15 KB	JAVA File	7/27/2005 12:36 PM

Figure 14. Directory generatemesh.

Section 3.5 includes an example that modifies, compiles, and runs file `Create2DMesh.java`. Java comments, near the start of each file, explain the file's use.

The logic for partition assignment in these files is very simple; elements are equally divided according to the number of requested partitions. Depending on the problem configuration, this may not give optimal parallel performance (see Section 7.4). Directory `generatemesh` also contains file `CreateHexPartitions.java`, which can be used to generate partitions for meshes composed of hexahedra; its use is explained by Java comments.

4.1.3. Periodic Boundary Conditions

CartaBlanca has a built-in optional mechanism to apply a periodic boundary condition to a computational mesh (essentially, the mesh can "wrap-around" itself). One reason to use a periodic boundary condition is to avoid artificial surface effects that can arise in a finite computational domain. Also, one of the code's periodic boundary condition options can be used to compute 3-D sections of a cylinder. Periodic boundary conditions are specified with the GUI **Physics** panel, using the booleans (checkboxes) **PeriodicInX**, **PeriodicInY**, **PeriodicInZ**, and **PeriodicInTheta** (see Section 4.2). For example, if **PeriodicInX** is selected, then the ends of the region in the *x*-direction are set as internal nodes by the code, and all pairs of partner nodes which are the periodic boundary nodes are determined. All fluxes going into the two nodes of the pair are added together and assigned to the pair. At the final state, each of the partner nodes should have the same physical values, which is implemented by averaging the values of the partner nodes and then assigning the average to them.

PeriodicInTheta is for a 3-D `cartesian` problem such as a cylinder in which one is only computing on a section (e.g., a quarter section); it imposes periodicity in the azimuthal angle. The problem is in Cartesian coordinates and neither *x*, *y*, or *z* is equivalent to the theta coordinate. One of the Cartesian axes is specified as an axis of rotation (using the **axisOfRotation** field in the **Physics** Panel). CartaBlanca will automatically find the appropriate periodic pairs.

CartaBlanca's general boundary conditions are described in Section 4.7. A problem can use a periodic boundary condition with a general boundary condition, as indicated in two of the following examples.

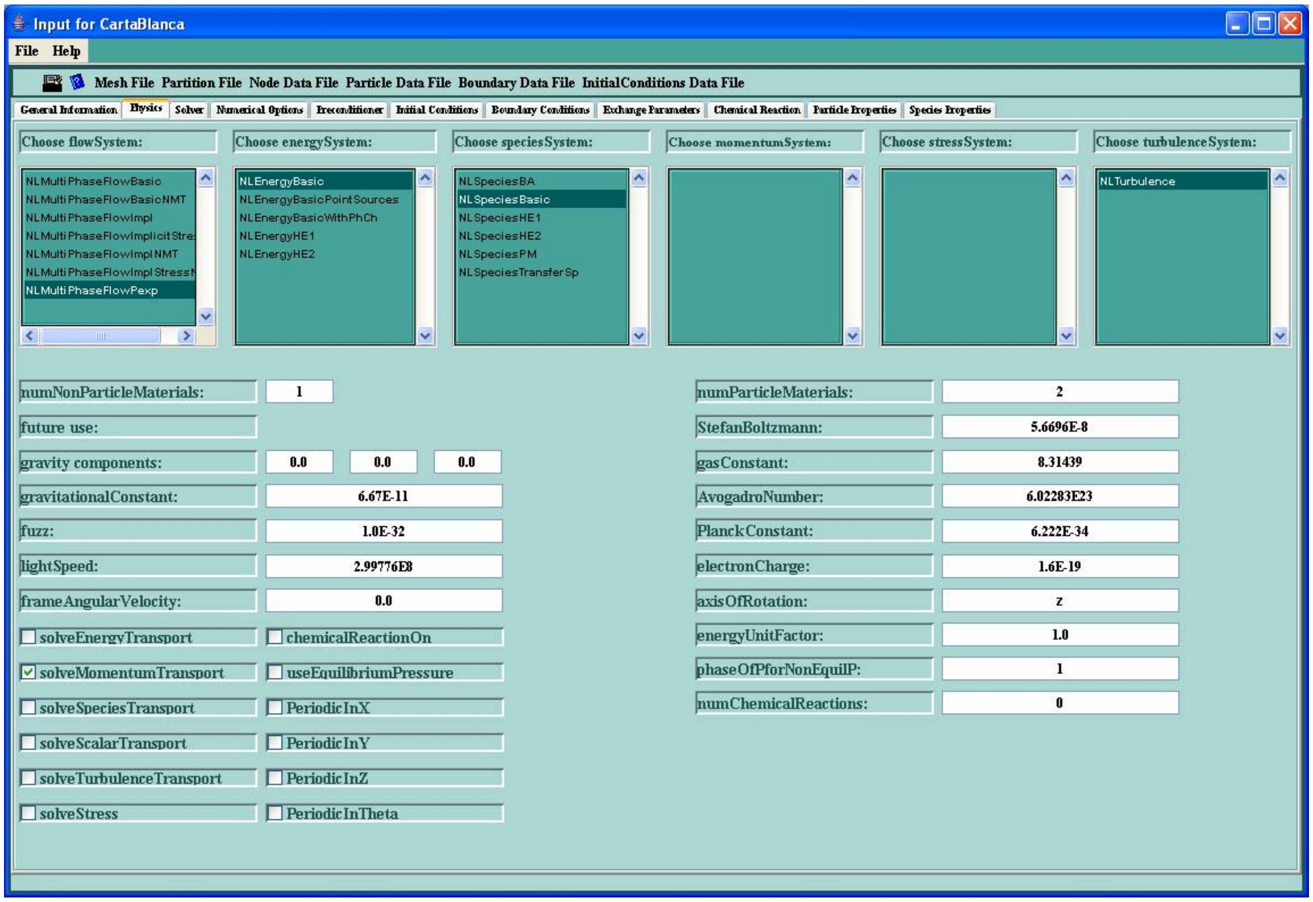
Examples: periodic boundary conditions: The CartaBlanca Test Suite contains seven problems that use a periodic boundary condition. For example,

- `testLongVibration1d.IO` is a 1-D `cartesian` problem that uses **PeriodicInX**.
- `Couette.IO` is a 2-D `cartesian` problem that uses **PeriodicInY**; `external wall` boundary conditions (see Section 4.7) are specified for velocities at the *x*-axis bounds.
- `DisOpsWithPeriodicity.IO` is a 3-D `cartesian` problem that uses **PeriodicInX**.
- `PoiseuilleCylind.IO` is a 2-D `cylindrical` problem that uses **PeriodicInY**; an `external reflective` boundary condition is applied at the `cylindrical` symmetry axis (x = 0), and an `external wall` boundary condition is applied for velocity specification at the end of the x-axis (see Section 4.7).
- `DisOpsWPInTheta.IO` is a 3-D `cartesian` problem that uses **PeriodicInTheta**.

4.2. Physics

The **Physics** panel is used to specify the physical processes to be modeled and the CartaBlanca algorithms to be used for their solution, such as selection of momentum transport and choice of a flow system for its solution. Also, supporting data such as physical constants are entered, and periodic boundary conditions (Section 4.1.3) can be selected.

Figure 15 shows the **Physics** panel. At the top of the panel there are six windows that can provide choices for solution algorithms ("solution systems", e.g., **Choose flowSystem**). The six windows allow choices for flow, energy, species, momentum, stress, and turbulence systems. The CartaBlanca Programmer's Manual shows a simple way to add new Java classes to the currently available choices (note that some of the current systems were developed for special cases). Below the solution-system windows are fields that are used to provide the physics parameters, and a set of checkboxes for selection of physics options (including the physical processes to be modeled).


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information | Physics | Solver | Numerical Options | Preconditioner | Initial Conditions | Boundary Conditions | Exchange Parameters | Chemical Reaction | Particle Properties | Species Properties

Choose flowSystem:
- NLMultiPhaseFlowBasic
- NLMultiPhaseFlowBasicNMT
- NLMultiPhaseFlowImpl
- NLMultiPhaseFlowImplicitStre
- NLMultiPhaseFlowImplNMT
- NLMultiPhaseFlowImplStressN
- NLMultiPhaseFlowPexp

Choose energySystem:
- NLEnergyBasic
- NLEnergyBasicPointSources
- NLEnergyBasicWithPhCh
- NLEnergyHE1
- NLEnergyHE2

Choose speciesSystem:
- NLSpeciesBA
- NLSpeciesBasic
- NLSpeciesHE1
- NLSpeciesHE2
- NLSpeciesPM
- NLSpeciesTransferSp

Choose momentumSystem:

Choose stressSystem:

Choose turbulenceSystem:
- NLTurbulence

numNonParticleMaterials:	1	numParticleMaterials:	2
future use:		StefanBoltzmann:	5.6696E-8
gravity components:	0.0 0.0 0.0	gasConstant:	8.31439
gravitationalConstant:	6.67E-11	AvogadroNumber:	6.02283E23
fuzz:	1.0E-32	PlanckConstant:	6.222E-34
lightSpeed:	2.99776E8	electronCharge:	1.6E-19
frameAngularVelocity:	0.0	axisOfRotation:	z
		energyUnitFactor:	1.0
		phaseOfPforNonEquilP:	1
		numChemicalReactions:	0

☐ solveEnergyTransport	☐ chemicalReactionOn
✓ solveMomentumTransport	☐ useEquilibriumPressure
☐ solveSpeciesTransport	☐ PeriodicInX
☐ solveScalarTransport	☐ PeriodicInY
☐ solveTurbulenceTransport	☐ PeriodicInZ
☐ solveStress	☐ PeriodicInTheta

Figure 15. GUI "Physics" tab.

The currently available solution algorithms (systems) are (where the prefix "NL" indicates "non-linear"):

Flow System

Note: To use a Flow System, the **Physics** panel boolean **solveMomentumTransport** must be checked.

NLMultiPhaseFlowBasic: Solves the momentum equation implicitly in pressure; the treatment of velocity is explicit. In the Newton-Krylov solver, the residual is basically the sum of volume fractions minus one. This class is used with implicit solvers, such as Gmres (see **Solver** panel description, Section 4.3).

NLMultiPhaseFlowBasicNMT: similar to NLMultiPhaseFlowBasic, used for a specific application.

NLMultiPhaseFlowImpl: Implicit time-advancement solution of the momentum equation. Currently under development.

NLMultiPhaseFlowImplicitStress: Implicit time-advancement solution of the momentum equation. Currently under development.

NLMultiPhaseFlowImplNMT: similar to NLMultiPhaseFlowImpl, used for a specific application.

NLMultiPhaseFlowImplStressNMT: similar to NLMultiPhaseFlowImplicitStress, used for a specific application.

NLMultiPhaseFlowPexp: Solves the momentum equation explicitly with solver NLExplicit (see **Solver** panel description, Section 4.3).

Note: Currently, NLMultiPhaseFlowPexp must be selected for problems that use the particle MPM/PIC method.

ThinPlate: solves deformation of thin plates under static or dynamic loads; temporary location in GUI to use this logic.

Energy System

Note: To use an Energy System, the **Physics** panel boolean **solveEnergyTransport** must be checked.

NLEnergyBasic: The basic system for solving the energy equation; can be used with implicit solvers or the NLExplicit solver (Section 4.3). However, even if NLExplicit is used, the treatment of the coupling term and the flux boundary condition term are still implicit.

NLEnergyBasicPointSources: Special application of the energy equation.

NLEnergyBasicWithPhCh: Extension of NLEnergyBasic with treatment for phase changes.

NLEnergyHE1: Special application of the energy equation for high explosive simulations with chemical reactions.

NLEnergyHE2: Special application of the energy equation for high explosive simulations with chemical reactions.

Species System

Note: To use a Species System, the **Physics** panel boolean **solveSpeciesTransport** must be checked.

`NLSpeciesBasic`: The basic system for solving the species equation.

`NLSpeciesBA`: Special application of the species equation.

`NLSpeciesHE1`: Special application of the species equation for high explosive simulations with chemical reactions.

`NLSpeciesHE2`: Special application of the species equation for high explosive simulations with chemical reactions.

`NLSpeciesPM`: Special application of the species equation.

`NLSpeciesTransferSp`: Extension of `NLSpeciesBasic` with species mass exchange.

Momentum System

This is a placeholder; currently there are no choices for this window (see **Flow System**).

Stress System

This is a placeholder; currently there are no choices for this window. Section 4.11 and the Theory Manual describe the material models available in CartaBlanca

Turbulence System

Note: To use the available Turbulence System, the **Physics** panel boolean **solveTurbulenceTransport** must be checked.

`NLTurbulence`: CartaBlanca uses the K-epsilon model for turbulent flow, where transport equations are solved for the turbulent kinetic energy (K) and the turbulent dissipation (ε).

The bottom of the **Physics** panel contains fields and checkboxes that are used to set values of physics-related parameters (including physical constants) and to select physics-related options (including physical processes to model). As indicated in Section 4.0.1, any units system can be used for the physical constants. These fields and checkboxes are as follows.

numParticleMaterials: integer; the number of materials (phases) to be modeled with the MPM/PIC method. A total of four materials (phases) is allowed, including non-particle materials (see **numNonParticleMaterials**). A material (phase) can comprise more than one species. Sections 4.10 and 4.11 describe input for the particle and material/species properties, respectively. The **Particles On** checkbox in the **General Information** panel must be checked if **numParticleMaterials** > 0.

numNonParticleMaterials: integer; the number of materials (phases) to be modeled with the ALE method. A total of four materials (phases) is allowed (**numNonParticleMaterials + numParticleMaterials** <= 4). A material (phase) can comprise more than one species.

gravity components: 3 reals, default values = 0.0; the components of the gravitational acceleration.

gravitationalConstant, lightSpeed, StefanBoltzmann, gasConstant, AvogadroNumber, PlanckConstant, electronCharge: 7 reals; self-explanatory. Any units system can be used (of course, it must be consistent with the rest of the model).

Note: The units of the universal gas constant and the activation energy in the Arrhenius chemical reaction term must be consistent; default value for the gas constant is 8.31439 J/mole-K (see **Chemical Reaction** panel description, Section 4.9).

fuzz: real; currently not used.

frameAngularVelocity: real; used for the angular velocity of an optional rotating coordinate system, for which the user must specify also the rotation axis; see **Physics** panel field **axisOfRotation**. The unit of angular velocity is so chosen that the resulting velocity is consistent with the unit of velocity.

axisOfRotation: keyword text; used in three cases, (1) for a **PeriodicInTheta** periodic boundary condition, to determine the x, y, z-axis, (2) for a cylindrical coordinate system, to determine the radial dimension, and (3) for a rotating coordinate frame, to determine how to add the centrifugal and Coriolis forces. Either `x`, `y`, or `z` (lower case).

Note: The **Physics** panel's **PeriodicInTheta** boolean must be checked to use that periodic boundary condition.

Note: Cylindrical coordinates are chosen with the **coordinateSystem** field in the **General Information** panel (Section 4.1).

Note: For a rotating coordinate system, the **Physics** panel parameter **frameAngularVelocity** must be set.

energyUnitFactor: real; used to make the terms pressure-dot and viscous dissipation in the energy equation consistent with other terms. The **energyUnitFactor** parameter was used during early code development, and is now obsolete; a value of 1.0 should always be used.

useEquilibriumPressure: boolean; if checked, the equilibrium pressure is used in the multiphase flow calculations. See also **Physics** panel field **phaseOfPforNonEquilP**.

phaseOfPforNonEquilP: integer; used for non-equilibrium pressure model (**Physics** panel checkbox **useEquilibriumPressure** is not checked). Specifies the phase (material) to use for the auxiliary pressure, as discussed in Chapter 3 of the Theory Manual. The phase to use is normally the fluid phase in the case of fluid-solid interactions. If only fluid phases exist, the equilibrium pressure model can be used. Here the phase number starts from 1.

solveEnergyTransport: boolean; if checked, energy equation is solved. One may also want to choose a suitable Energy System for the problem; otherwise, the Energy System currently in the input file is used.

solveMomentumTransport: boolean; if checked, momentum equation is solved. One may also want to choose a suitable Flow System for the problem; otherwise, the Flow System currently in the input file is used.

solveSpeciesTransport: boolean; if checked, species transport equation is solved. One may also want to choose a suitable Species System for the problem; otherwise, the Species System currently in the input file is used.

solveScalarTransport: boolean; if checked, a special scalar transport physics system is solved. This is in a test developed in the early days of the code.

solveTurbulenceTransport: boolean; if checked, the K-epsilon model will be used, with Turbulence System `NLTurbulence`, which must be specified in the input. See also input parameters **turbK** and **turbE** in the **Initial Conditions** panel (Section 4.6), and **tkH**, **tkPhi**, **tkFl**, **tlH**, **tlPhi**, and **tlFl** in the **Boundary Conditions** panel (Section 4.7).

solveStress: boolean; if checked, the code will solve for stress on the Eulerian grid; otherwise the code will only solve stress on MPM/PIC particles. Typically, particles are used to solve for stress in solid materials, and before the **solveStress** option was implemented, particles were the only choice for a solid phase. With the **solveStress** option, we do not need particles if we prefer only to use the Eulerian grid to solve the problem.

Note: The **solveStress** parameter does not use the **Physics** panel's "Choose stressSystem:" window; it is planned to change its name to **solveStressOnGrid**.

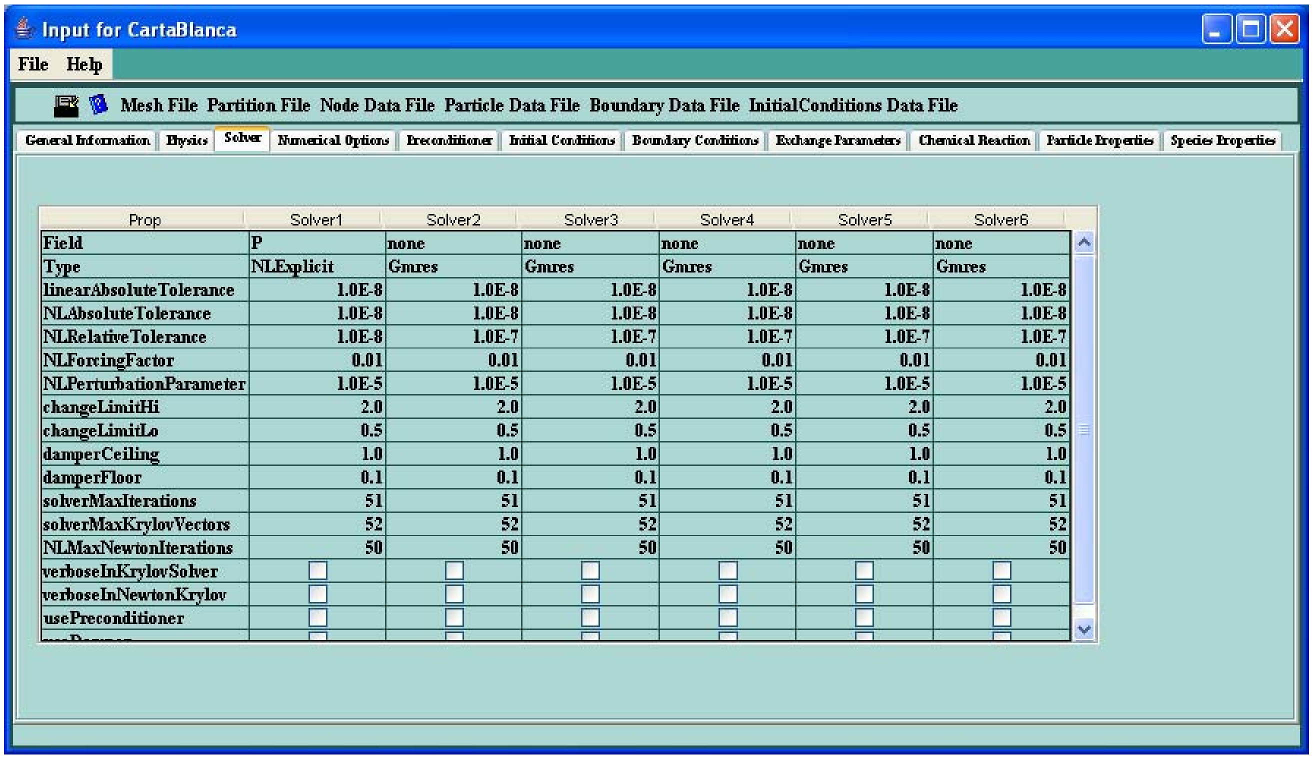
chemicalReactionOn: boolean; if checked, the chemical reaction model will be used. Input specifications for modeling chemical reactions are described in Section 4.9 ("Chemical Reaction"). See also **Physics** panel field **numChemicalReactions**.

numChemicalReactions: integer; the number of chemical reactions to be modeled. See **Physics** panel checkbox **chemicalReactionOn**.

PeriodicInX, PeriodicInY, PeriodicInZ, PeriodicInTheta: 4 booleans; if checked, a periodic boundary condition is used for the indicated coordinate. See Section 4.2.1 for a description of CartaBlanca's periodic boundary conditions.

4.3. Solver

The **Solver** panel is used to specify and configure the solvers of the field equations used by the model. All data are entered in a single scrollable table. Figure 16 shows the startup display of the **Solver** panel's table. The first column identifies the various properties (parameters) for each solver to be used. Up to six solvers may be used for a given model; the data for each solver are entered in each of the next six columns ("Solver1" ….. "Solver6"). The solver data are comprised of text fields (keywords), reals, integers, and booleans (checkboxes). Keyword entry is facilitated by selection from dropdown lists.



Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information | Physics | Solver | Numerical Options | Preconditioner | Initial Conditions | Boundary Conditions | Exchange Parameters | Chemical Reaction | Particle Properties | Species Properties

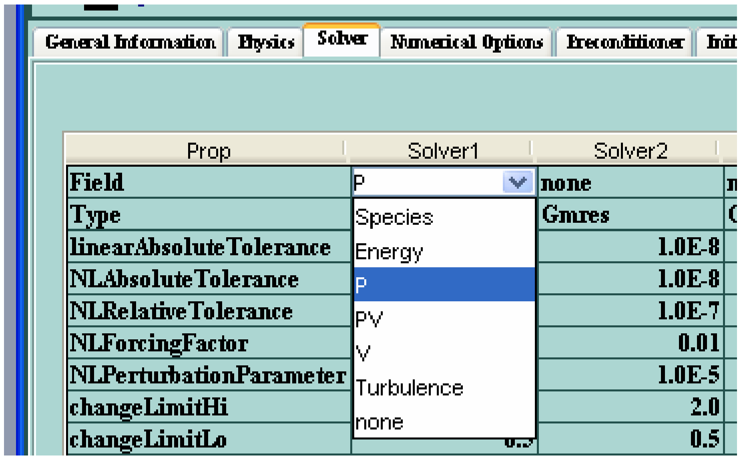
Prop	Solver1	Solver2	Solver3	Solver4	Solver5	Solver6
Field	P	none	none	none	none	none
Type	NLExplicit	Gmres	Gmres	Gmres	Gmres	Gmres
linearAbsoluteTolerance	1.0E-8	1.0E-8	1.0E-8	1.0E-8	1.0E-8	1.0E-8
NLAbsoluteTolerance	1.0E-8	1.0E-8	1.0E-8	1.0E-8	1.0E-8	1.0E-8
NLRelativeTolerance	1.0E-8	1.0E-7	1.0E-7	1.0E-7	1.0E-7	1.0E-7
NLForcingFactor	0.01	0.01	0.01	0.01	0.01	0.01
NLPerturbationParameter	1.0E-5	1.0E-5	1.0E-5	1.0E-5	1.0E-5	1.0E-5
changeLimitHi	2.0	2.0	2.0	2.0	2.0	2.0
changeLimitLo	0.5	0.5	0.5	0.5	0.5	0.5
damperCeiling	1.0	1.0	1.0	1.0	1.0	1.0
damperFloor	0.1	0.1	0.1	0.1	0.1	0.1
solverMaxIterations	51	51	51	51	51	51
solverMaxKrylovVectors	52	52	52	52	52	52
NLMaxNewtonIterations	50	50	50	50	50	50
verboseInKrylovSolver	☐	☐	☐	☐	☐	☐
verboseInNewtonKrylov	☐	☐	☐	☐	☐	☐
usePreconditioner	☐	☐	☐	☐	☐	☐

Figure 16. GUI "Solver" tab.

The input parameters for the **Solver** panel are as follows:

Field: keyword text; the field to be solved, either `Species`, `Energy`, `P`, `PV`, `V`, `Turbulence`, or `none`, where `P` is pressure and `V` is velocity. In case of more than one solver, the order must be `P`, `Energy`, and `Species`. The code will indicate an error and abort if the fields are specified in a different order. Also, the code will point out an error and abort if an inconsistency is detected (for example, checking **solveEnergyTransport** in the **Physics** panel and not assigning an energy solver).

Figure 17 shows selection of a solver's **Field** using the panel table's built-in dropdown list for that parameter.

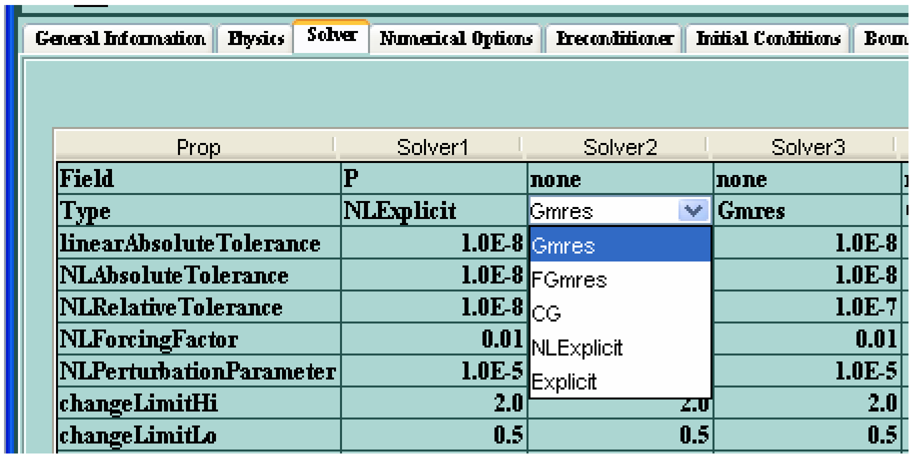


Prop	Solver1	Solver2
Field	P	none
Type		Gmres
linearAbsoluteTolerance		1.0E-8
NLAbsoluteTolerance		1.0E-8
NLRelativeTolerance		1.0E-7
NLForcingFactor		0.01
NLPerturbationParameter		1.0E-5
changeLimitHi		2.0
changeLimitLo		0.5

Figure 17. Selecting a solver field.

Type: keyword text; the solver algorithm, either `Gmres`, `FGmres`, `CG`, `NLExplicit`, or `Explicit`, where `Gmres` is the Generalized Minimum RESidual method, `FGmres` is the Flexible Generalized Minimum RESidual method, CG is the Conjugate Gradient method, `NLExplicit` is the NonLinear EXPLICIT method, and `Explicit` is the EXPLICIT method. Each of these solver methods corresponds to a Java class in the `solver` package. These are the available methods for users to choose to solve the resulting algebraic equations after numerical discretization. The optimal choice depends on the specific physical problem.

Figure 18 shows selection of a solver's **Type** using the panel table's built-in dropdown list for that parameter. (Of course a **Field** must also be selected, and the **Field**s' order must be correct.)



Prop	Solver1	Solver2	Solver3
Field	P	none	none
Type	NLExplicit	Gmres	Gmres
linearAbsoluteTolerance	1.0E-8		1.0E-8
NLAbsoluteTolerance	1.0E-8		1.0E-8
NLRelativeTolerance	1.0E-8		1.0E-7
NLForcingFactor	0.01		0.01
NLPerturbationParameter	1.0E-5		1.0E-5
changeLimitHi	2.0	2.0	2.0
changeLimitLo	0.5	0.5	0.5

Figure 18. Selecting a solver type.

linearAbsoluteTolerance: real; tolerance in the solver.

NLAbsoluteTolerance: real; absolute tolerance in the Newton-Krylov solver.

NLRelativeTolerance: real; relative tolerance in the Newton-Krylov solver.

NLForcingFactor: real; will be the relative tolerance in the specified solvers.

NLPerturbationParameter: real; currently not used.

The following four variables, **changeLimitHi**, **changeLimitLo**, **damperCeiling**, and **damperFloor**, are used in the calculation of the damping coefficient in the Newton-Krylov solver. They allow increasing the radius of convergence for an initial guess by employing a damped iteration (see [7]). See also the **useDamper** checkbox in this panel.

changeLimitHi: real; used in the calculation of the damping coefficient in the Newton-Krylov solver.

changeLimitLo: real; used in the calculation of the damping coefficient in the Newton-Krylov solver..

damperCeiling: real; used in the calculation of the damping coefficient in the Newton-Krylov solver.

damperFloor: real; used in the calculation of the damping coefficient in the Newton-Krylov solver.

solverMaxIterations: integer; maximum number of solver iterations.

solverMaxKrylovVectors: integer; maximum number of vectors in the Newton-Krylov solver.

NLMaxNewtonIterations: integer; maximum number of Newton-Krylov iterations.

verboseInKrylovSolver: boolean; if checked, print details on the convergence of the solver at each iteration.

verboseInNewtonKrylov: boolean; if checked, print details on the convergence at each Newton-Krylov iteration.

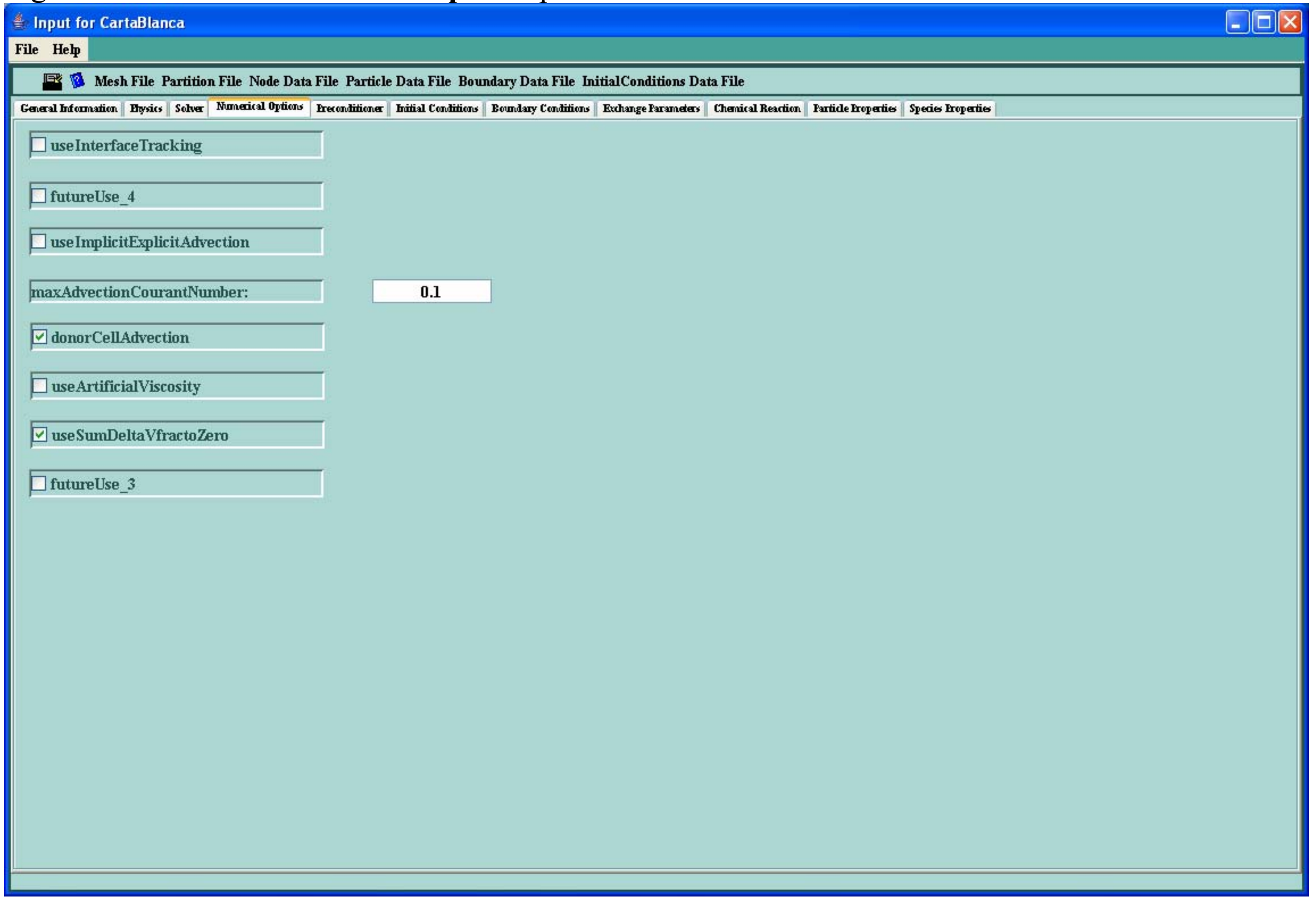
usePreconditioner: boolean; if checked, use the preconditioner specified in the **Preconditioner** panel (see Section 4.5).

useDamper: boolean; if checked, use the damper. In this case, it is necessary to specify the four variables in this panel used in the calculation of the damping coefficient.

4.4. Numerical Options

The **Numerical Options** panel contains parameters that affect the behavior of the ALE and MPM/PIC numerics. There are three checkboxes (and four others that are currently not used) and one field for a real.

Figure 19 shows the **Numerical Options** panel.


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties

☐ useInterfaceTracking

☐ futureUse_4

☐ useImplicitExplicitAdvection

maxAdvectionCourantNumber: 0.1

☑ donorCellAdvection

☐ useArtificialViscosity

☑ useSumDeltaVfractoZero

☐ futureUse_3

Figure 19. GUI "Numerical Options" tab.

The input parameters for the **Numerical Options** panel are as follows:

<u>useInterfaceTracking</u>: boolean; currently not used.

<u>futureUse_4</u>: boolean; currently not used.

<u>useImplicitExplicitAdvection</u>: boolean; currently not used.

<u>maxAdvectionCourantNumber</u>: real, default value = 0.5; Courant number is defined as speed$\times \Delta t / \Delta x$, where the speed is the maximum of the material speeds of the all the phases and the speed of sound for explicit calculations. A smaller Courant number usually results in a more stable calculation but causes longer run time.

donorCellAdvection: boolean; if checked, the upwind scheme will be used to advect transport quantities across the computational cells. This scheme is quite diffusive but often provides stability to a calculation.

useArtificialViscosity: boolean; if checked, CartaBlanca adds artificial viscosity terms to the pressure on cell surfaces. Its use is recommended to damp unphysical oscillations for shock wave problems. However, its use causes unphysical dissipation and therefore is not recommended for problems not involving strong discontinuities.

useSumDeltaVfractionZero: boolean; currently we suggest this box be checked whenever the MPM/PIC method is used and unchecked whenever the MPM/PIC method is not used.

Note: It is planned to remove the option **useSumDeltaVfractionZero** from the user input in a future code version.

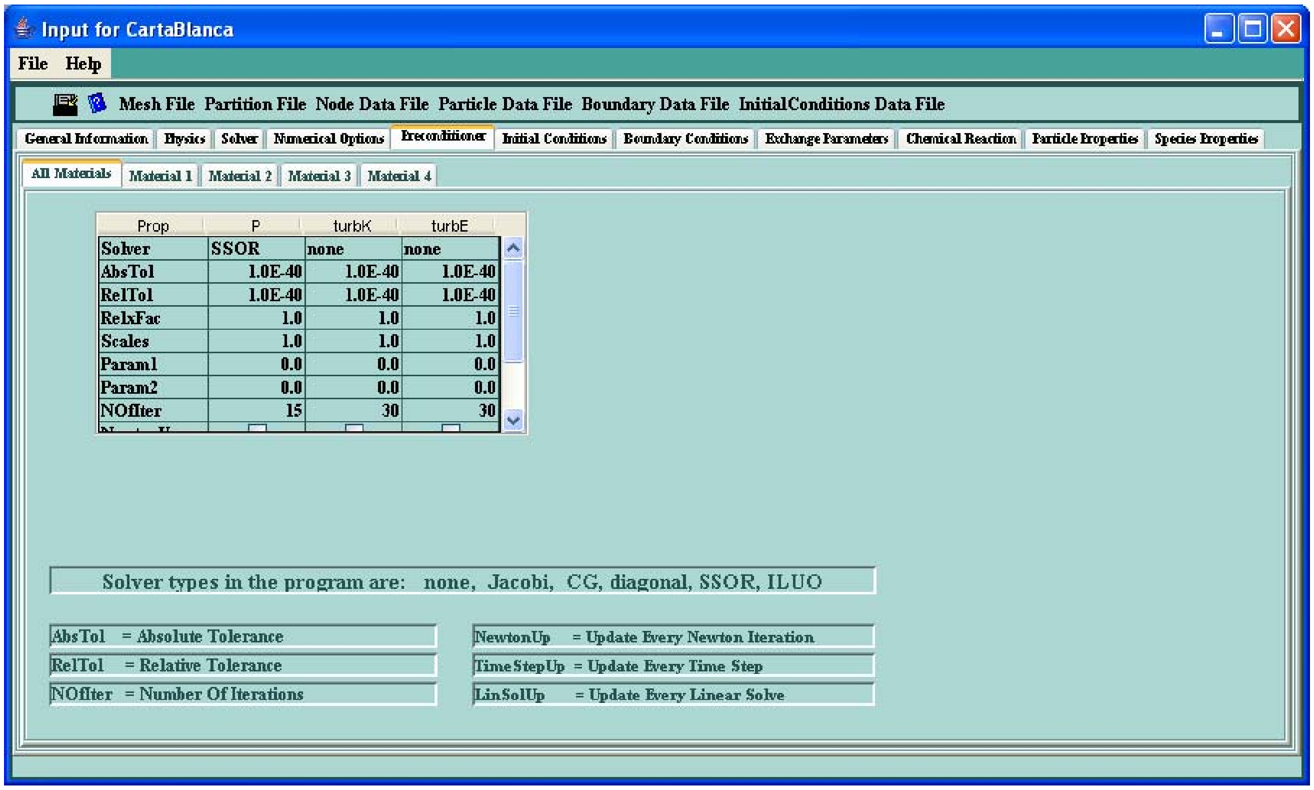
futureUse_3: boolean; currently not used.

4.5. Preconditioner

The **Preconditioner** panel is used to specify and configure the Newton-Krylov preconditioners. The Theory Manual gives a detailed discussion on CartaBlanca's preconditioning logic. The **Preconditioner** panel is composed of five sub-panels: **All Materials**, **Material 1**, **Material 2**, **Material 3**, and **Material 4**. Sub-panel **All Materials** is used for preconditioning quantities that are common to all materials in the model. The other sub-panels are used for quantities that are specific to a material (phase), where up to four materials can be defined in a given model (in the **Preconditioner** panel, input is provided for up to four species per material).

The data for a sub-panel are entered in a scrollable table for that sub-panel. The first column of each table lists the parameters (properties) of the preconditioners. These parameters are the same in all five tables. The remaining columns are for data entry for each preconditioner; the quantity to be preconditioned appears in the column heading. The preconditioner data are comprised of text fields (keywords), reals, integers, and booleans (checkboxes). Keyword entry is facilitated by selection from dropdown lists.

Figure 20 shows the **All Materials** (sub-)panel, at the top of the panel's table. Figure 21 shows the same table scrolled to the bottom. Columns 2, 3, and 4 are for entry of data for pressure (P) and the K-ε turbulence model (turbK and turbE).

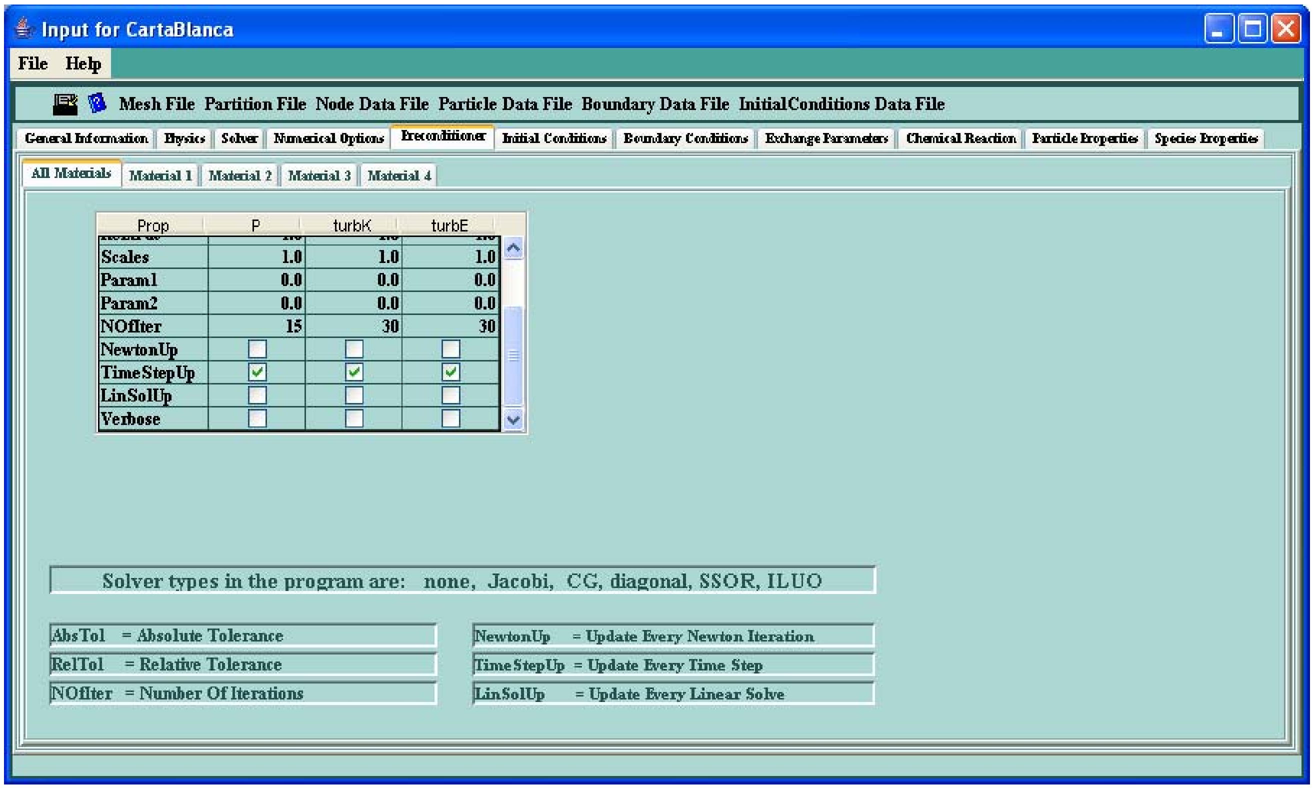

Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
All Materials Material 1 Material 2 Material 3 Material 4

Prop	P	turbK	turbE
Solver	SSOR	none	none
AbsTol	1.0E-40	1.0E-40	1.0E-40
RelTol	1.0E-40	1.0E-40	1.0E-40
RelxFac	1.0	1.0	1.0
Scales	1.0	1.0	1.0
Param1	0.0	0.0	0.0
Param2	0.0	0.0	0.0
NOfIter	15	30	30

Solver types in the program are: none, Jacobi, CG, diagonal, SSOR, ILUO

AbsTol = Absolute Tolerance
RelTol = Relative Tolerance
NOfIter = Number Of Iterations
NewtonUp = Update Every Newton Iteration
TimeStepUp = Update Every Time Step
LinSolUp = Update Every Linear Solve

Figure 20. GUI "Preconditioner" tab, "All Materials" sub-tab.



Prop	P	turbK	turbE
Scales	1.0	1.0	1.0
Param1	0.0	0.0	0.0
Param2	0.0	0.0	0.0
NOfIter	15	30	30
NewtonUp	☐	☐	☐
TimeStepUp	✓	✓	✓
LinSolUp	☐	☐	☐
Verbose	☐	☐	☐

Solver types in the program are: none, Jacobi, CG, diagonal, SSOR, ILUO

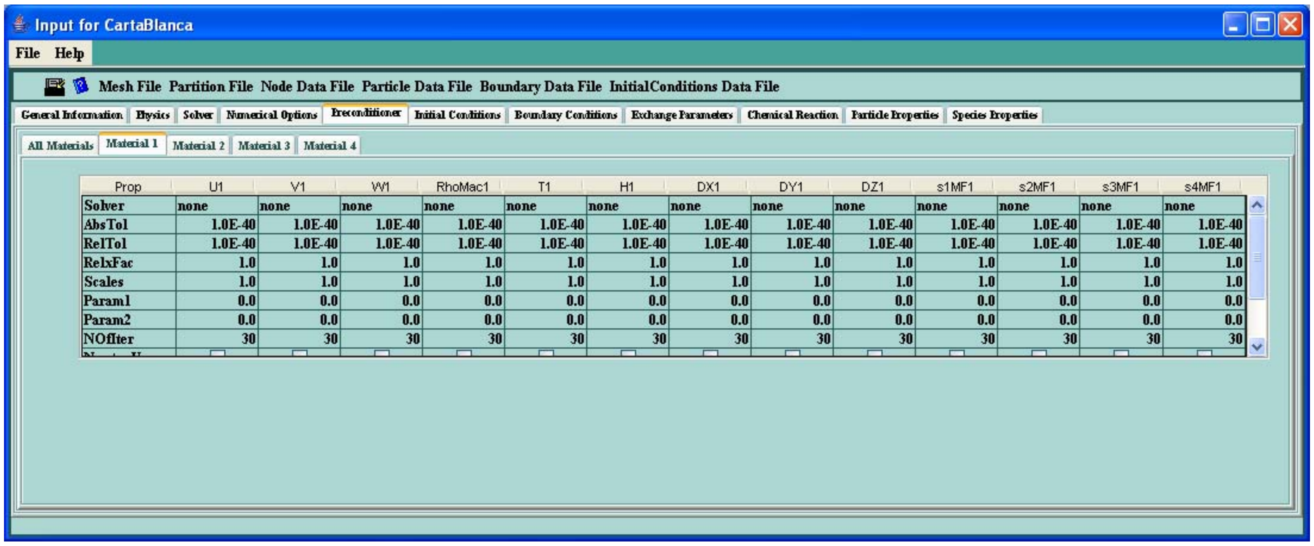
AbsTol = Absolute Tolerance
RelTol = Relative Tolerance
NOfIter = Number Of Iterations
NewtonUp = Update Every Newton Iteration
TimeStepUp = Update Every Time Step
LinSolUp = Update Every Linear Solve

Figure 21. Preconditioner "All Materials" sub-tab, bottom of table.

Figure 22 shows the **Material 1** panel.

The material-specific quantities for which a preconditioner can be specified are, for Materialn (where n can be 1, 2, 3, or 4):

Un,	X component of the velocity,
Vn,	Y component of the velocity,
Wn,	Z component of the velocity,
RhoMacn,	macroscopic density,
Tn,	temperature,
Hn,	enthalpy,
DXn,	X component of the displacement,
DYn,	Y component of the displacement,
DZn,	Z component of the displacement,
s1MFn,	mass fraction of species 1 in Materialn,
s2MFn,	mass fraction of species 2 in Materialn,
s3MFn,	mass fraction of species 3 in Materialn, and
s4MFn,	mass fraction of species 4 in Materialn.



Prop	U1	V1	W1	RhoMac1	T1	H1	DX1	DY1	DZ1	s1MF1	s2MF1	s3MF1	s4MF1
Solver	none	none	none	none	none	none	none	none	none	none	none	none	none
AbsTol	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40
RelTol	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40	1.0E-40
RelxFac	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Scales	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Param1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Param2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
NOfIter	30	30	30	30	30	30	30	30	30	30	30	30	30

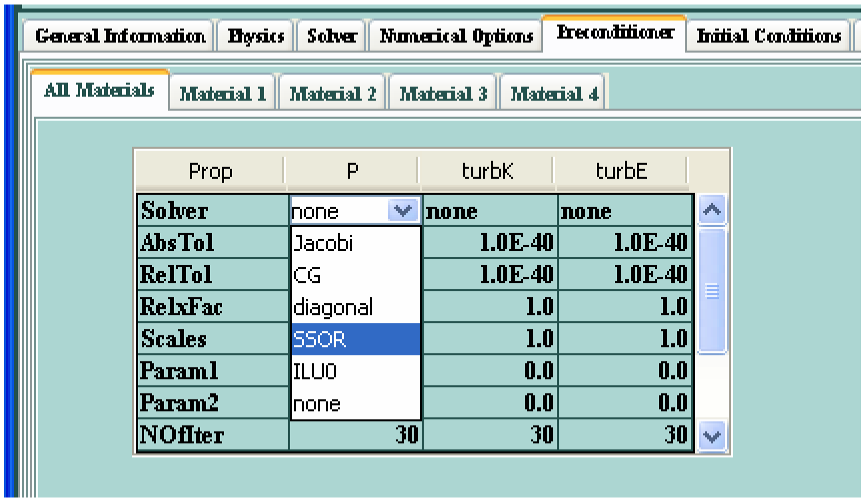
Figure 22. Preconditioner "Material 1" sub-tab.

The input parameters (properties) for each preconditioned quantity on each sub-panel of the **Preconditioner** panel are as follows:

<u>**Solver**</u>: keyword text; the preconditioner solution algorithm, either `Jacobi`, `CG`, `diagonal`, `SSOR`, `ILU0`, or `none`. The optimal choice depends on the specific physical problem. These methods are implemented in the Java `solver` package.

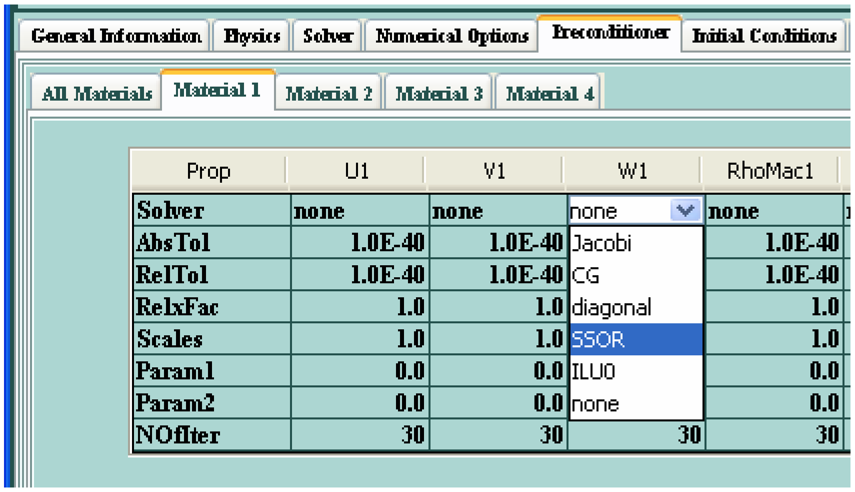
Figure 23 shows selection of the pressure preconditioner solver from a built-in dropdown list. Figure 24 shows selection of the Material 1 velocity (Z component) preconditioner solver from a built-in dropdown list.

<u>Note</u>: The SSOR and ILU0 methods cannot be used for parallel calculations. The code will write an error message and shut down in this circumstance.



Prop	P	turbK	turbE
Solver	none	none	none
AbsTol		1.0E-40	1.0E-40
RelTol		1.0E-40	1.0E-40
RelxFac		1.0	1.0
Scales		1.0	1.0
Param1		0.0	0.0
Param2		0.0	0.0
NOfIter	30	30	30

Figure 23. Selecting a preconditioner solver for pressure.



Prop	U1	V1	W1	RhoMac1
Solver	none	none	none	none
AbsTol	1.0E-40	1.0E-40		1.0E-40
RelTol	1.0E-40	1.0E-40		1.0E-40
RelxFac	1.0	1.0		1.0
Scales	1.0	1.0		1.0
Param1	0.0	0.0		0.0
Param2	0.0	0.0		0.0
NOfIter	30	30	30	30

Figure 24. Selecting a preconditioner solver for Material 1 velocity (Z component).

<u>AbsTol</u>: real; absolute tolerance.

<u>RelTol</u>: real; relative tolerance.

<u>RelxFac</u>: real; currently not used.

<u>Scales</u>: real; value for scaling the preconditioner variable.

<u>Param1</u>: real; currently not used.

<u>Param2</u>: real; currently not used.

<u>NOfIter</u>: integer; number of iterations.

<u>NewtonUp</u>: boolean; if checked, update every Newton iteration.

<u>TimeStepUp</u>: boolean; if checked, update every time step.

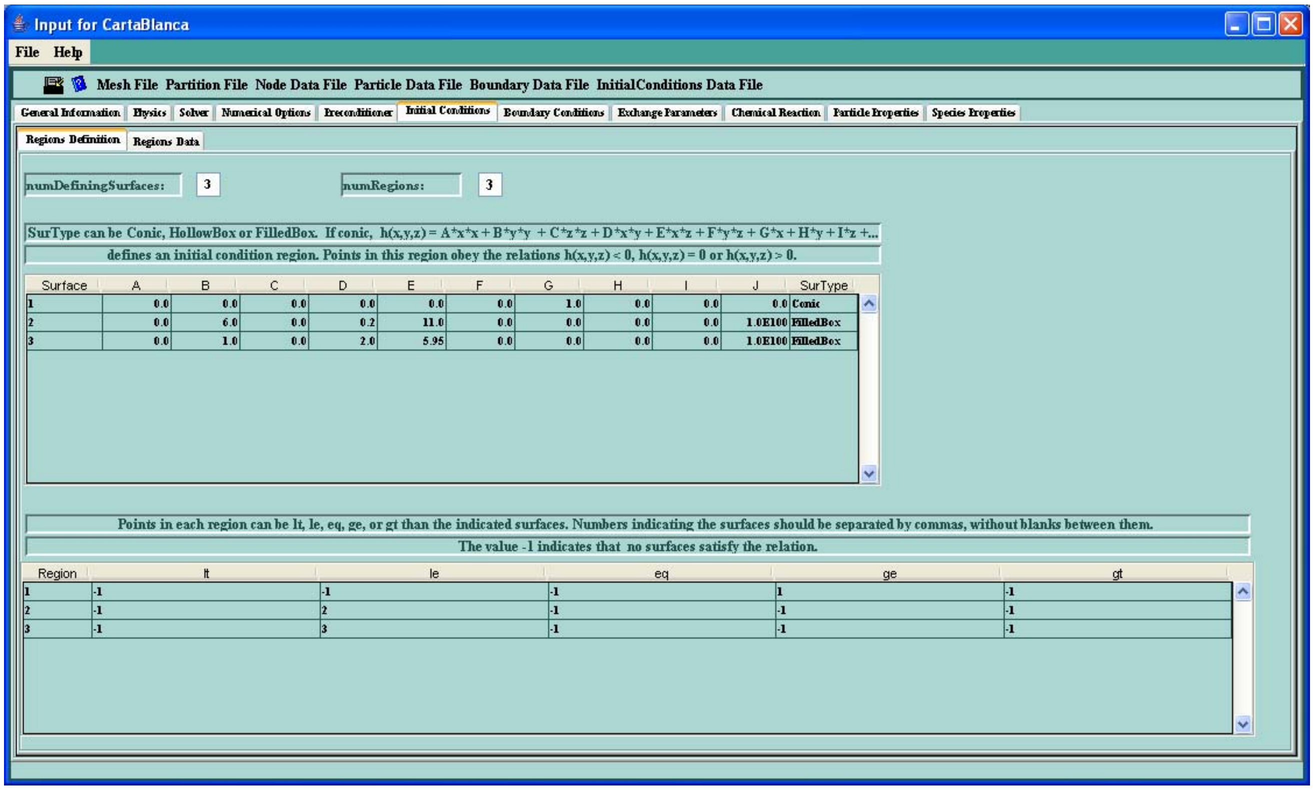
<u>LinSolUp</u>: boolean; if checked, update every linear solve.

<u>Verbose</u>: boolean; if checked, print detailed diagnostics (e.g., L2-norms).

4.6. Initial Conditions

The **Initial Conditions** panel is used to specify the initial locations of the problem's materials and their initial velocities, temperatures, etc. Also, the species mass fractions of the materials are specified here. The initial conditions are set by first breaking the problem domain up into geometric regions, and then assigning time = 0 properties to the regions. A region is defined by using one or more surfaces in the problem's one-, two-, or three-dimensional space. The **Initial Conditions** panel consists of two sub-panels: **Regions Definition**, where the surfaces and regions are defined, and **Regions Data**, where the region properties are specified. Note that the layout of the **Boundary Conditions** panel (Section 4.7) is very similar to that of the **Initial Conditions** panel.

Section 3.5 gives an example of using the **Regions Definition** panel to set up surfaces and regions for a 2-D problem. Figure 25 shows the **Regions Definition** panel; it contains two tables and two integer fields that set the number of rows in the tables. The upper table is used to define the surfaces and the lower table defines the regions by referring to the surfaces. At the top of the panel, the **numDefiningSurfaces** field sets the number of rows in the upper (surface) table. If **numDefiningSurfaces** is 0 (the default), the surface table will be empty. Otherwise, the code automatically fills in the Surface id number(s) in the table's first column. Three types of surface are available: `Conic`, `HollowBox`, and `FilledBox`. The surface type is entered in the last (12th) column of the table. Figure 26 shows selection of the type of Surface 3 from a built-in dropdown list.


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
Regions Definition Regions Data

numDefiningSurfaces: 3 numRegions: 3

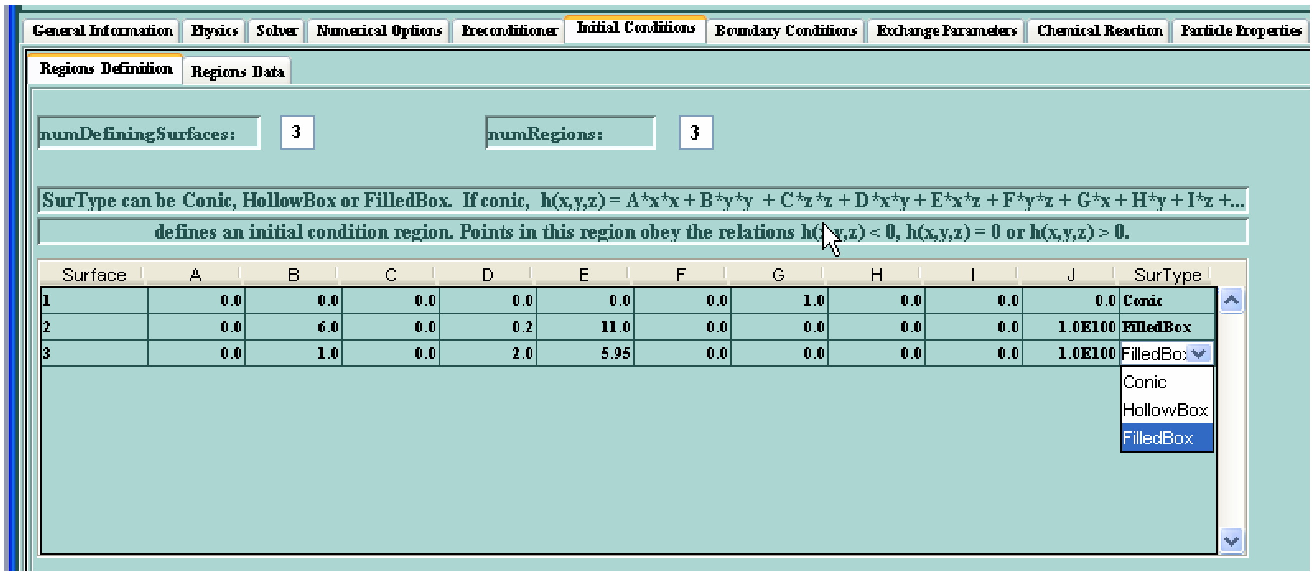
SurType can be Conic, HollowBox or FilledBox. If conic, h(x,y,z) = A*x*x + B*y*y + C*z*z + D*x*y + E*x*z + F*y*z + G*x + H*y + I*z +...
defines an initial condition region. Points in this region obey the relations h(x,y,z) < 0, h(x,y,z) = 0 or h(x,y,z) > 0.

Surface	A	B	C	D	E	F	G	H	I	J	SurType
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	Conic
2	0.0	6.0	0.0	0.2	11.0	0.0	0.0	0.0	0.0	1.0E100	FilledBox
3	0.0	1.0	0.0	2.0	5.95	0.0	0.0	0.0	0.0	1.0E100	FilledBox

Points in each region can be lt, le, eq, ge, or gt than the indicated surfaces. Numbers indicating the surfaces should be separated by commas, without blanks between them.
The value -1 indicates that no surfaces satisfy the relation.

Region	lt	le	eq	ge	gt
1	-1	-1	-1	1	-1
2	-1	2	-1	-1	-1
3	-1	3	-1	-1	-1

Figure 25. GUI "Initial Conditions" tab ("Regions Definition" sub-tab).



numDefiningSurfaces: 3   numRegions: 3

SurType can be Conic, HollowBox or FilledBox. If conic, h(x,y,z) = A*x*x + B*y*y + C*z*z + D*x*y + E*x*z + F*y*z + G*x + H*y + I*z +...
defines an initial condition region. Points in this region obey the relations h(x,y,z) < 0, h(x,y,z) = 0 or h(x,y,z) > 0.

Surface	A	B	C	D	E	F	G	H	I	J	SurType
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	Conic
2	0.0	6.0	0.0	0.2	11.0	0.0	0.0	0.0	0.0	1.0E100	FilledBox
3	0.0	1.0	0.0	2.0	5.95	0.0	0.0	0.0	0.0	1.0E100	FilledBox

Figure 26. Setting initial conditions: selecting a surface type for Surface 3.

For `Conic` surfaces, the 2nd through 11th columns are used to enter coefficients A,…,J to define the surface:

$$h(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J \ .$$

The `FilledBox` option is used to define a rectangular parallelepiped with opposite vertices (x_1, y_1, z_1) and (x_2, y_2, z_2), by specifying A = x_1, B = y_1, C = z_1, D = x_2, E = y_2, and F = z_2. In this case, G, H, I, and J are not used. The `HollowBox` option is used to define a rectangular parallelepiped with an inner parallelepiped inside it. The values of A through F are defined exactly in the same way as for the `FilledBox`, and the variable G is used for defining the distance between the outer box and the inner box. `FilledBox` and `HollowBox` surfaces are used in the regions table in a manner similar to `Conic` surfaces, as described below (where the inner box and the outer box together define a `HollowBox`). In a 2-D problem, the variables C and F are not used and should be set to zero.

The field **numRegions** specifies the number of initial regions; the code creates the appropriate number of rows in the bottom (regions) table, and fills in the Region id number(s). In each row, we indicate how the region is positioned with respect to the surfaces by entering Surface id numbers in columns 2 – 6, which specify the relations lt, le, eq, ge, and gt, respectively ("less than", etc.). The `Conic` surfaces *h(x,y,z)* are used in the regions table to define regions in which the points obey one or more of the relations $h(x,y,z) < 0.0$, $h(x,y,z) \le 0.0$, $h(x,y,z) = 0.0$, $h(x,y,z) \ge 0.0$, $h(x,y,z) > 0.0$. One or more Surface id numbers are entered in the appropriate columns to obtain the desired relations (see Section 3.5). With `FilledBox` and `HollowBox` surfaces, the columns labeled lt, le, ge, and gt are also used to define a region, where "lt" indicates a point is in the region (as far as that surface is concerned), and the other relations have a corresponding sense ("eq" should not be used). The number –1 is used when a relation does not apply for any of the surfaces (i.e., no surface satisfies the relation). Surface types can be mixed in a given relation. If there is more than one surface in a relation, the id numbers must be separated by commas, with no embedded blanks. Note that, in the code, the checks for $h(x,y,z) \le 0.0$ and $h(x,y,z) \ge 0.0$ are actually done using $h(x,y,z) < \varepsilon$ and $h(x,y,z) > -\varepsilon$ respectively, where ε is a small tolerance, entered in the GUI as the **tolerance** field in the **BcDefinitions** sub-panel of the **Boundary Conditions** panel (Section 4.7). This is to account for possible error in the node data file (Section 4.1.1) due to round-off error. For example, in a 1D case, a node intended to have the value 0.3 may be printed as 0.300000000000000001 in the node data file but one does not notice this. Thus, if one wants to use an initial region $x \le 0.3$ by only checking $x \le 0.3$, the node will not be in the region, which is not the user's intention. Similarly, the check $h(x,y,z) = 0.0$ is done using $|h(x,y,z)| < \varepsilon$.

If all the entries are –1 for a region, the region is not defined by surfaces; the region's node coordinates must be provided in an input text file that is specified by the **General Information** panel's field **InitialConditionsFileName**. An example of such a file is:

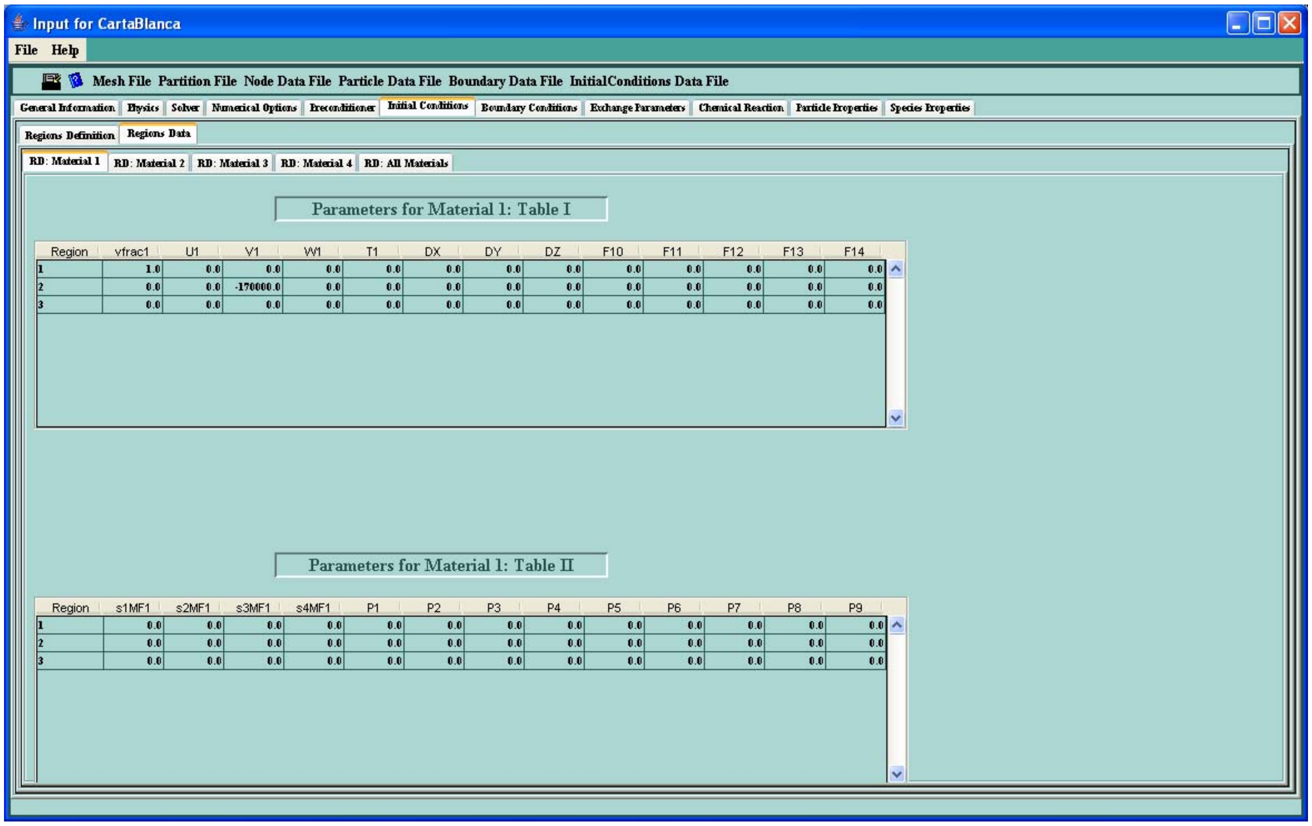
```
2
0 753
0.3 2.5 0.0
0.3 2.75 0.0
0.3 3.0 0.0
0.3 3.25 0.0
0.3 3.5 0.0
………
………
………
```

where the first line gives the number of initial regions specified in the file (2 in this case), the second line specifies the index of the first initial region in the file (starting from 0) and the number of computational (mesh) nodes in this initial region (753 here), and the next 753 lines give the x, y, z coordinates of the nodes. A second data block gives the next index and node count, and the node coordinates. Note that the code, internally, starts the region indexing at 0 (Java arrays start at 0), but the GUI presents region ids that start at 1.

The initial regions can be overlapping; if a node appears in two or more initial regions, the initial condition specified in the region with the largest index is in effect. The initialization proceeds from initial region 0 to the highest-numbered one. For each initial region, the code loops through all the nodes in the whole region to determine if a node is in this initial region, and if it is, the code then assigns initial values to the node according to the initial data of this initial region, thus, overwriting the node's current values.

The **Regions Data** panel is used to initialize each region. **Regions Data** comprises five sub-panels: **RD: Material 1**, **RD: Material 2**, **RD: Material 3**, **RD: Material 4**, and **RD: All Materials**. The first four of these panels are used to enter material (phase)-dependent data, and the fifth is used for material-independent data.

Figure 27 shows **Regions Data -- RD: Material 1**. Each material panel has two tables. The upper table accepts, for each region, the initial value of the volume fraction, the initial values of the components of the velocity (U, V, and W), the initial value of the temperature, and the initial values of the components of the displacement (DX, DY, and DZ). The last five columns (F10 through F14) are not used and serve as placeholders for future variables.



Parameters for Material 1: Table I

Region	vfrac1	U1	V1	W1	T1	DX	DY	DZ	F10	F11	F12	F13	F14
1	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	-170000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

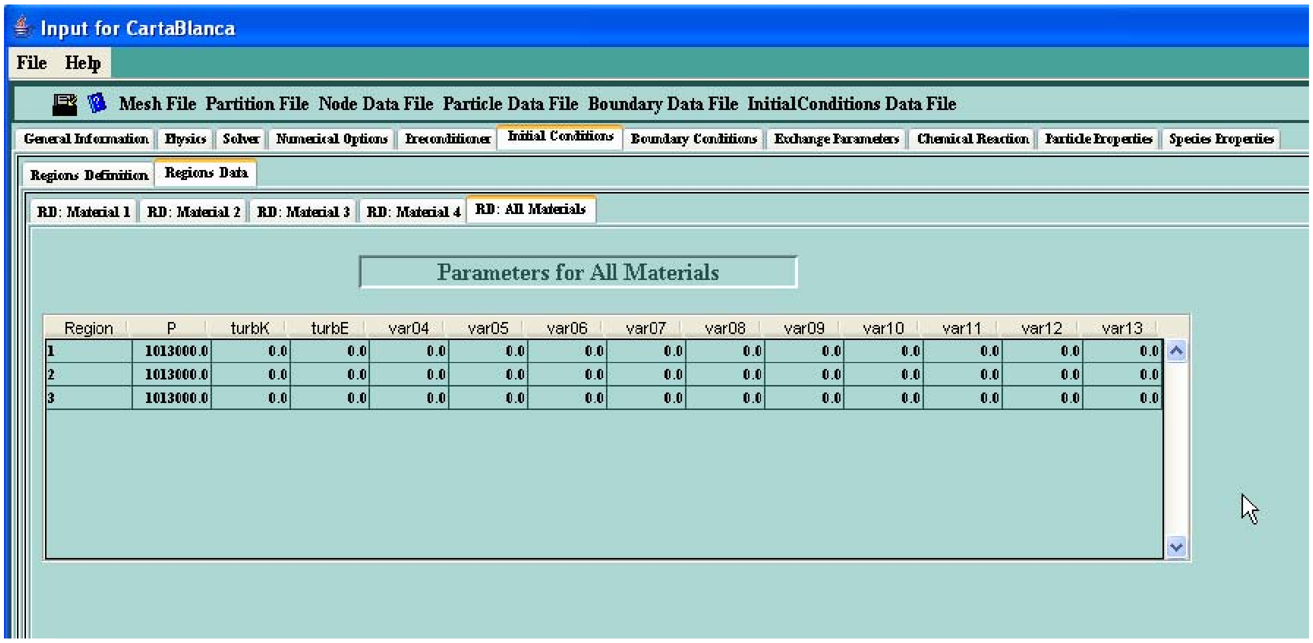
Parameters for Material 1: Table II

Region	s1MF1	s2MF1	s3MF1	s4MF1	P1	P2	P3	P4	P5	P6	P7	P8	P9
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 27. Initial Conditions "Regions Data" sub-tab ("RD: Material 1" sub-tab).

The lower table accepts, for each region, the species mass fractions for up to four species per material. The last nine columns (P1 through P9) are not used and serve as placeholders for future variables.

Figure 28 shows **Regions Data -- RD: All Materials**. There is a single table that accepts, for each region, the initial values of the pressure (P) and the input parameters for the K-epsilon turbulence model (turbK and turbE). The last 10 columns (var04,…,var13) are not used and serve as placeholders for future variables.


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
Regions Definition Regions Data
RD: Material 1 RD: Material 2 RD: Material 3 RD: Material 4 RD: All Materials

Parameters for All Materials

Region	P	turbK	turbE	var04	var05	var06	var07	var08	var09	var10	var11	var12	var13
1	1013000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1013000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1013000.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 28. Initial Conditions "Regions Data" sub-tab ("RD: All Materials" sub-tab).

The input parameters for the **Initial Conditions** panel are as follows:

Regions Definition panel

numDefiningSurfaces: integer; number of surfaces that will be used to specify the initial regions.

numRegions: integer; number of initial regions.

SurType: keyword text; the surface type, either `Conic`, `FilledBox`, or `HollowBox`. Entered for each surface in the surface table.

A, B, C, D, E, F, G, H, I, J: ten reals; one set of values for each surface. For **SurType** `Conic`, the coefficients of the 3-D conic surface
$h(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J$.
For **SurType** `FilledBox`, the opposite vertices (x_1, y_1, z_1) and (x_2, y_2, z_2) of a rectangular parallelepiped, where A = x_1, B = y_1, C = z_1, D = x_2, E = y_2, and F = z_2 (G, H, I, and J are not used). For **SurType** `HollowBox`, the opposite vertices (x_1, y_1, z_1) and (x_2, y_2, z_2) of an outer rectangular parallelepiped, where A = x_1, B = y_1, C = z_1, D = x_2, E = y_2, and F = z_2, and G is the distance from the outer rectangular parallelepiped to an inner rectangular parallelepiped (H, I, and J are not used). For both `FilledBox` and `HollowBox`, in a 2-D problem, C and F are not used and should be set to zero.

lt, le, eq, ge, gt: five text fields; one set of entries for each region. For **SurType** `Conic`, the surface id numbers are entered to specify the relations $h(x,y,z) < 0.0$, $h(x,y,z) = 0.0$, or $h(x,y,z) > 0.0$. For **SurType** `FilledBox` and **SurType** `HollowBox`, the surface id numbers are used to specify if a point is in the box, where "lt" has the sense of including the point, "gt" is used to exclude the point, etc. ("eq" should not be used). For `Conic`, `FilledBox`, and `HollowBox` surfaces, where more than one surface is used for a relation, the id numbers must be separated by commas, with no embedded blanks; if a relation does not apply for a region (i.e., no surface satisfies the relation), "-1" should be entered. Surface types may be mixed in a given relation. If all five relations have "-1" for a region, a file that gives the region's nodes must be provided (see discussion above in this section, and Section 4.1).

Regions Data panel

RD: Material 1, **RD: Material 2**, **RD: Material 3**, and **RD: Material 4** panels

vfrac1, **vfrac2**, **vfrac3**, or **vfrac4**: real; for each region, initial volume fraction of Material 1, 2, 3, or 4.

U1, **U2**, **U3**, or **U4**: real; for each region, initial X-component of velocity of Material 1, 2, 3, or 4.

V1, **V2**, **V3**, or **V4**: real; for each region, initial Y-component of velocity of Material 1, 2, 3, or 4.

W1, **W2**, **W3**, or **W4**: real; for each region, initial Z-component of velocity of Material 1, 2, 3, or 4.

T1, **T2**, **T3**, or **T4**: real; for each region, initial temperature of Material 1, 2, 3, or 4.

DX, **DY**, and **DZ**: three reals; for each region, initial X, Y, and Z component of displacement of Material 1, 2, 3, or 4.

F10, **F11**, **F12**, **F13**, and **F14**: five reals; currently not used.

s1MF1, **s1MF2**, **s1MF3**, or **s1MF4**: real; for each region, initial species-number 1 mass fraction in Material 1, 2, 3, or 4.

s2MF1, **s2MF2**, **s2MF3**, or **s2MF4**: real; for each region, initial species-number 2 mass fraction in Material 1, 2, 3, or 4.

s3MF1, **s3MF2**, **s3MF3**, or **s3MF4**: real; for each region, initial species-number 3 mass fraction in Material 1, 2, 3, or 4.

s4MF1, **s4MF2**, **s4MF3**, or **s4MF4**: real; for each region, initial species-number 4 mass fraction in Material 1, 2, 3, or 4.

P1, **P2**, **P3**, **P4**, **P5**, **P6**, **P7**, **P8**, and **P9**: nine reals; currently not used.

RD: All Materials panel

P: real; for each region, initial pressure.

turbK: real; for each region, initial K-parameter (turbulent kinetic energy) for K-epsilon turbulence model.

turbE: real; for each region, initial ε-parameter (turbulent dissipation) for K-epsilon turbulence model.

var04, **var05**, **var06**, **var07**, **var08**, **var09**, **var10**, **var11**, **var12**, and **var13**: ten reals; currently not used.

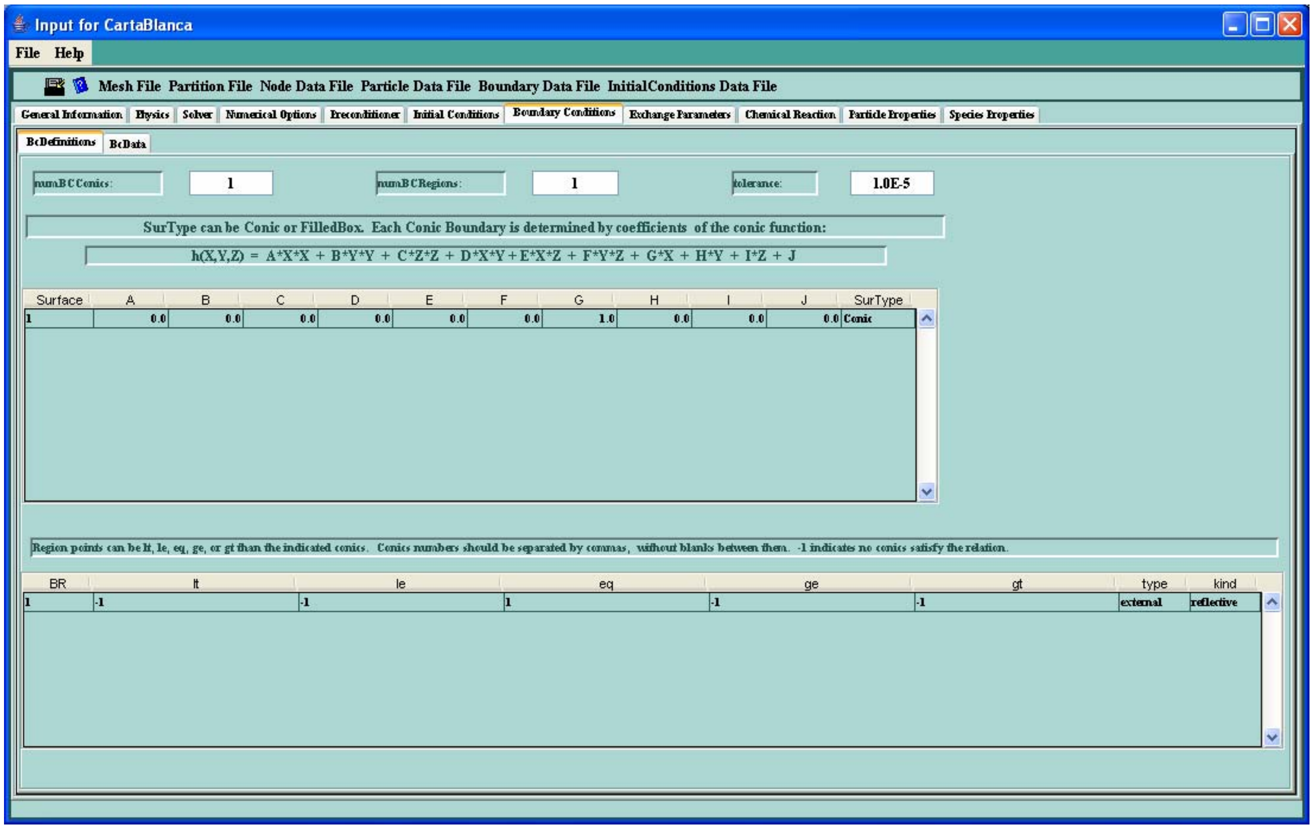
4.7. Boundary Conditions

This section describes the setup of computational boundary conditions and the applicable boundary condition variables. CartaBlanca applies boundary conditions at computational nodes that are specified by the user. Typically, these nodes are on the outer border of the computational domain ("external" nodes), but the option is available to specify internal nodes for special situations. In either case, the boundary condition nodes can be specified by identifying geometric regions in the domain that contain the nodes, and/or by supplying a file that has node coordinates. Note that the GUI **Physics** panel (Section 4.2) is used to specify periodic boundary conditions (see also Section 4.1.3).

Boundary conditions are specified with the **Boundary Conditions** panel, which is very similar to the **Initial Conditions** panel (Section 4.6). There are two **Boundary Conditions** sub-panels: **BcDefinitions** and **BcData**. Figure 29 shows the **BcDefinitions** panel. **BcDefinitions** is used in the same manner as the **Initial Conditions** panel's **Regions Definition** panel: there are upper and lower tables that define surfaces and regions in the computational domain, respectively, where here the regions are used to apply specific boundary conditions. The numbers of surfaces and regions are specified with **numBCConics** and **numBCRegions**, respectively. In addition, the tolerance for accepting a point as part of a region is specified in the **tolerance** field. Again, if the surface type, **SurType**, is `Conic`, the coefficients in the upper table define one or more surfaces:

$$h(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J \ .$$

These surfaces are used to define regions in which the points obey one or more of the relations $h(x,y,z) < 0.0, h(x,y,z) \leq 0.0$, $h(x,y,z) = 0.0$, $h(x,y,z) \geq 0.0$, $h(x,y,z) > 0.0$. The code tests for $h(x,y,z) \leq 0.0$ and $h(x,y,z) \geq 0.0$ using $h(x,y,z) < \varepsilon$ and $h(x,y,z) > -\varepsilon$ respectively, where ε is a small value entered in the **tolerance** field. This is to account for possible error in the node data file (Section 4.1.1) due to round-off. For example, in a 1-D case, a node intended to have the value *x*=0.3 may be printed as 0.300000000000000001 in the node data file but one does not notice this. Thus, if one wants to use a boundary region $x \leq 0.3$, the node would not be in the region, an unintended result. Similarly, the check $h(x,y,z) = 0.0$ is done using $|h(x,y,z)| < \varepsilon$. The value of **tolerance** is also used for the **Initial Conditions** regions (Section 4.6).


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
BcDefinitions BcData

numBCConics: 1 numBCRegions: 1 tolerance: 1.0E-5

SurType can be Conic or FilledBox. Each Conic Boundary is determined by coefficients of the conic function:

h(X,Y,Z) = A*X*X + B*Y*Y + C*Z*Z + D*X*Y+E*X*Z + F*Y*Z + G*X + H*Y + I*Z + J

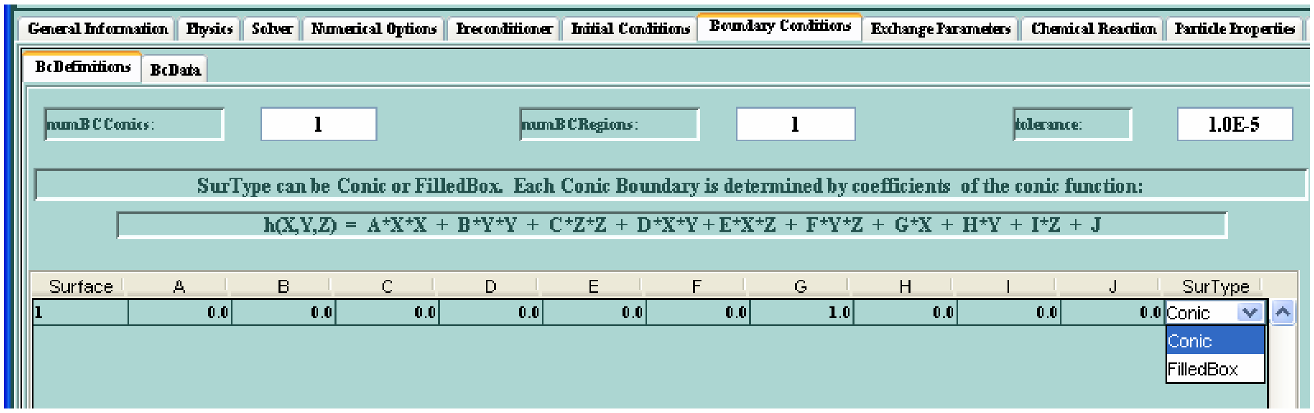
Surface	A	B	C	D	E	F	G	H	I	J	SurType
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	Conic

Region points can be lt, le, eq, ge, or gt than the indicated conics. Conics numbers should be separated by commas, without blanks between them. -1 indicates no conics satisfy the relation.

BR	lt	le	eq	ge	gt	type	kind
1	-1	-1	1	-1	-1	external	reflective

Figure 29. GUI "Boundary Conditions" tab ("BcDefinitions" sub-tab).

In addition to `Conic`, **SurType** can be `FilledBox`, as described in Section 4.6 (`HollowBox` is not allowed). Figure 30 shows the selection of a surface type from a built-in dropdown list.


General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties
BcDefinitions BcData
Conic
FilledBox

numBCConics: 1 numBCRegions: 1 tolerance: 1.0E-5

SurType can be Conic or FilledBox. Each Conic Boundary is determined by coefficients of the conic function:

h(X,Y,Z) = A*X*X + B*Y*Y + C*Z*Z + D*X*Y+E*X*Z + F*Y*Z + G*X + H*Y + I*Z + J

Surface	A	B	C	D	E	F	G	H	I	J	SurType
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	Conic

Figure 30. Selecting a boundary condition surface type.

In the lower table, for each of the regions the **type** of boundary condition (`internal` or `external`) and the **kind** of boundary condition (`wall`, `reflective`, `reflcorner`, `inflow`, `outflow`, `pressure`, `inflow-outflow`, or `vel-direction`) are entered. Figures 31 and 32 show the selection of a boundary region **type** and **kind**, respectively, from dropdown lists. The **type** indicates whether the region's nodes are to be treated by the code as on the outer (`external`) border of the mesh, or inside the mesh (`internal`). The eight **kind**s of boundary condition are used to specify different physical properties at the boundary condition nodes (e.g., the uses of `wall`, `inflow`, `outflow`, and `pressure` are apparent from their names). Descriptions of the **type** and **kind** options are given in Section 4.7.1. Again, as with the **Initial Conditions** panel, we indicate how each region is positioned with respect to each of the surfaces using the **lt**, **le**, **eq**, **ge**, and **gt** columns. The entry "–1" is used when a relation does not apply to any of the surfaces. The reader is referred to Section 4.6 for additional details on specifying surfaces and regions.

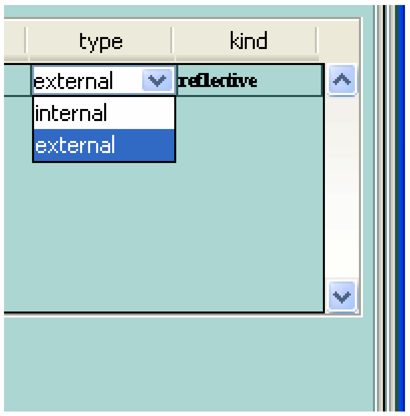


Figure 31. Selecting a boundary region type.

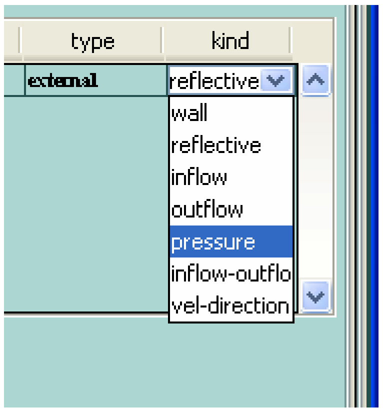


Figure 32. Selecting a boundary region kind.

If all the entries for a boundary-condition region are –1, the region is not defined by surfaces; the computational-node coordinates of the region must be provided in an input file that is specified in the **General Information** panel's **BoundaryFileName** field (Section 4.1). An example of such a file is:

```
3
0 pressure 1
1.2500000000000002 18.25 0.0
1 outflow 1
3.5 17.75 0.0
2 inflow 2
0.425 2.5 0.0
0.55 2.5 0.0
```

The first line gives the number of boundary regions specified in the file (3 in this case). Following are sets of data for each of the regions. The second line in the example specifies the index of the first boundary region in the file (0 in this case), the kind of the boundary region (pressure), and the number of nodes in the boundary region (1). The following line gives the X, Y, Z coordinates of the single node. This is followed by a 1-node outflow region and a 2-node inflow region. Note that, as with the initial conditions data file, the region indices start at 0. Also, the boundary condition "kind" specification in the file is currently not used (but a dummy entry must be made to allow parsing of the file); the actual kind is read from the GUI's region table. An example of the use of a boundary file is in the CartaBlanca Test Suite problem `testPoiseuille1_RF_Test`, which writes `.IO` file `Poiseulle1_RF.IO`, where boundary region 2 has "-1" for the items **lt**, **le**, **eq**, **ge**, and **gt**; this triggers the use of a boundary data file. In the **General Information** panel, **BoundaryFileName** has the file `meshes\2D\QUADS\Poiseuille\myBCFile.txt`, which has:

```
1
1 wall 2
10.0 0.0 0.0
10.0 1.0 0.0
```

where one boundary region is specified, with index 1 (indicating the second region in the problem), and coordinates for two nodes are given. Boundary files can be convenient when the region has a complex geometry.

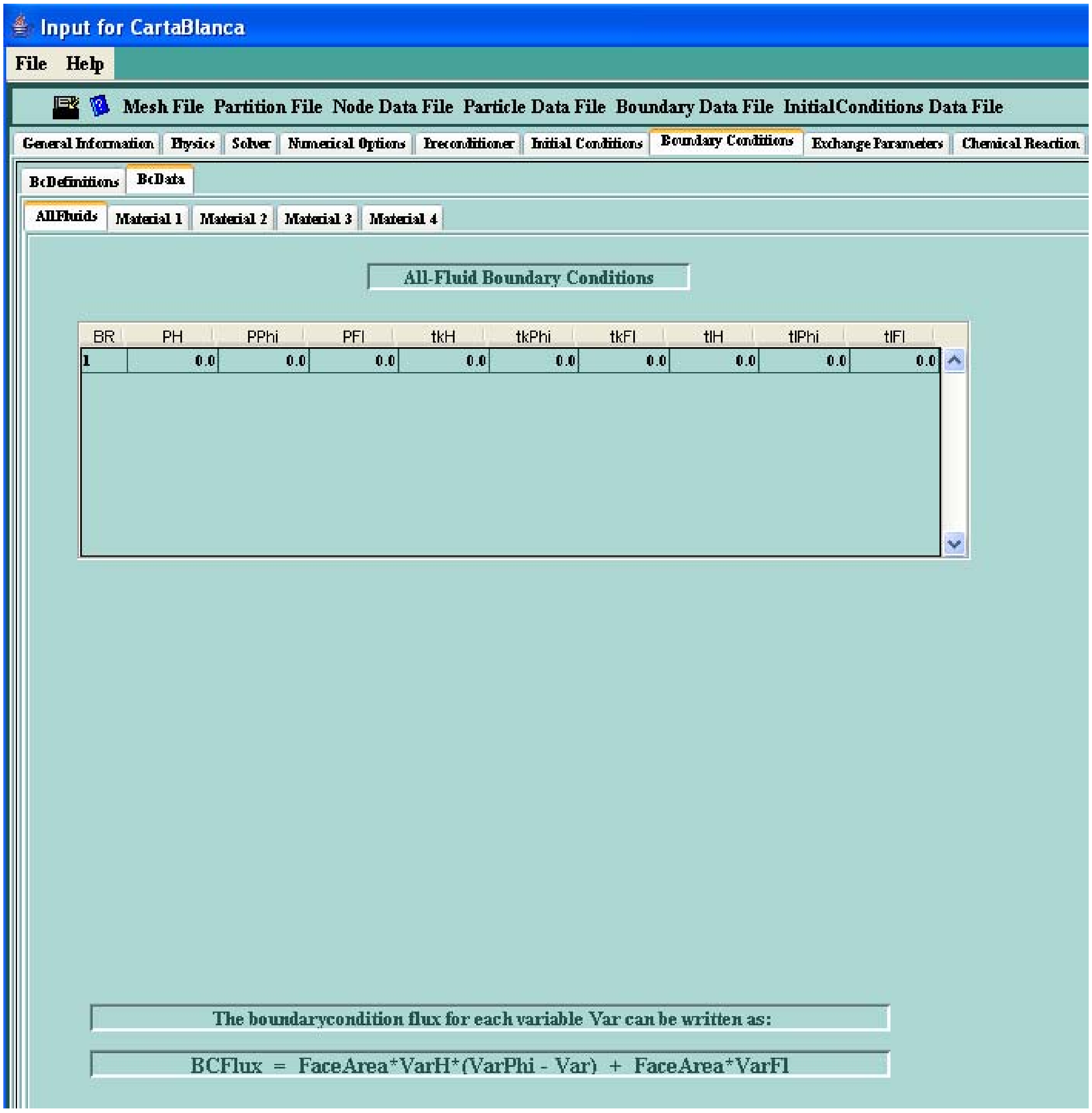
The boundaries are ordered by the boundary region number, followed by a possible default boundary (of **kind** `wall`), which includes all external nodes that do not belong to boundaries specified in the boundary region table (or as periodic). If a node has been claimed by a boundary, then it is out of consideration by the boundaries that follow; that is, a node cannot belong to more than one boundary.

The **BcData** panel is used to specify the boundary conditions in each boundary condition region. **BcData** comprises five sub-panels: **AllFluids**, **Material 1**, **Material 2**, **Material 3**, and **Material 4**. The **AllFluids** panel is used to enter boundary condition data that are common to all materials (phases) in the problem; the other four panels are for material (phase)-specific boundary condition data. In each of the five **BcData** panels, each of the boundary condition variables has three terms, identified by the suffixes **H**, **Phi** and **Fl**, that are used for calculating the variable's boundary flux (e.g., temperature Temp has input for **TempH**, **TempPhi**, and **TempFl**). Using these terms, the boundary condition flux of variable *Var* is defined as

$$BCFlux = FaceArea * VarH * (VarPhi - Var) + FaceArea * VarFl \ .$$

The use of the **H**, **Phi** and **Fl** parameters by the various boundary **kind**s is discussed in Section 4.7.1.

Figure 33 shows **BcData** -- **AllFluids**. There is a single table that accepts, for each boundary condition region (BR), data sets for pressure (P) and for the K-epsilon turbulence model (tk and tl) (Note: Currently the tk and tl cells are placeholders).


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction
BcDefinitions BcData
AllFluids Material 1 Material 2 Material 3 Material 4
All-Fluid Boundary Conditions

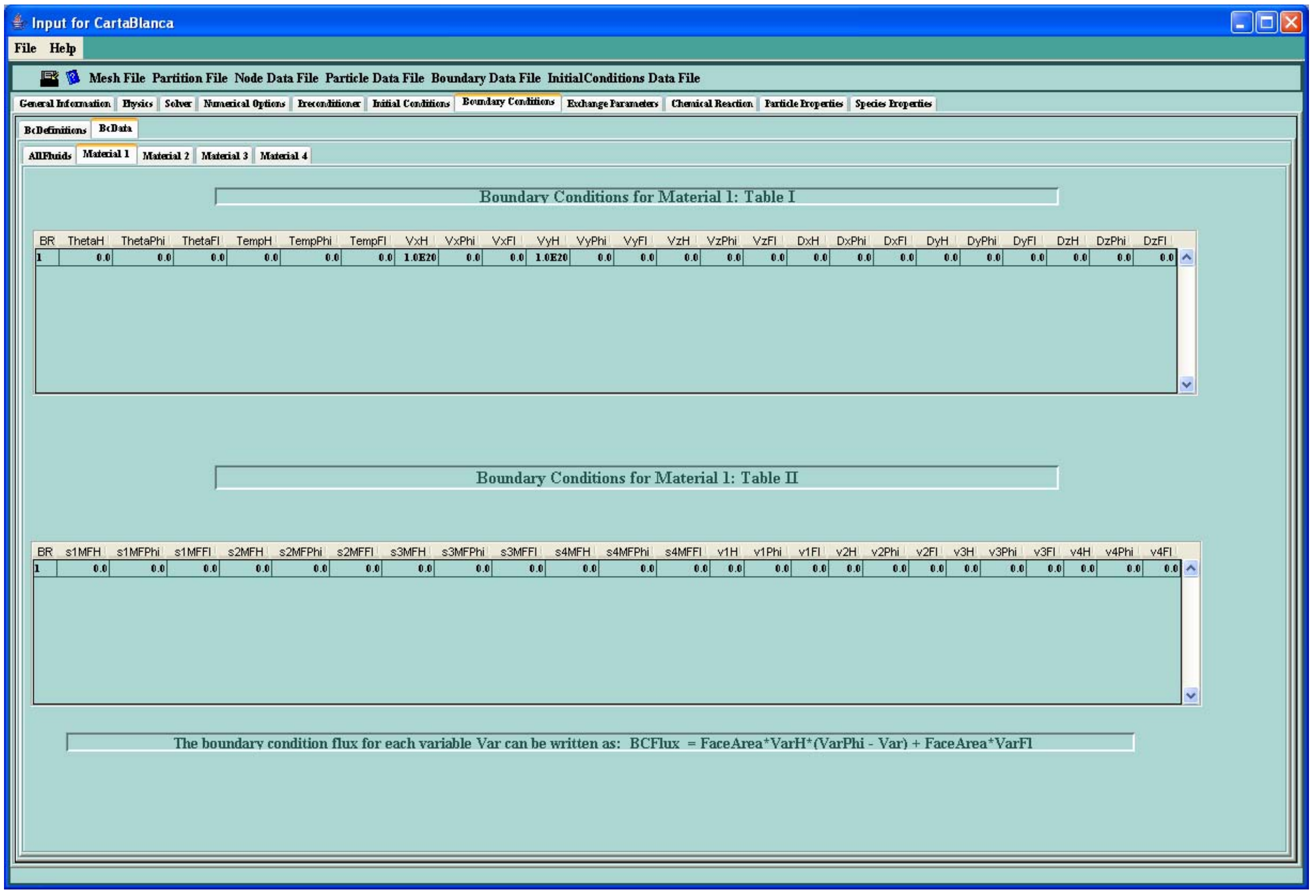
BR	PH	PPhi	PFl	tkH	tkPhi	tkFl	tlH	tlPhi	tlFl
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The boundarycondition flux for each variable Var can be written as:

BCFlux = FaceArea*VarH*(VarPhi - Var) + FaceArea*VarFl

Figure 33. GUI Boundary Conditions "BcData" sub-tab ("AllFluids" sub-tab).

Figure 34 shows **BcData** -- **Material 1**. Each material panel has two tables. The upper table accepts, for each boundary condition region (BR), data sets for volume fraction (Theta), temperature (Temp), velocity components (Vx, Vy, Vz), and displacement components (Dx, Dy, Dz). The lower table accepts, for each region, the species mass fractions for up to four species per material (s1MF, s2MF, s3MF, s4MF); variables v1, v2, v3, and v4 are currently unused.


Input for CartaBlanca

Boundary Conditions for Material 1: Table I

BR	ThetaH	ThetaPhi	ThetaFl	TempH	TempPhi	TempFl	VxH	VxPhi	VxFl	VyH	VyPhi	VyFl	VzH	VzPhi	VzFl	DxH	DxPhi	DxFl	DyH	DyPhi	DyFl	DzH	DzPhi	DzFl
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0E20	0.0	0.0	1.0E20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Boundary Conditions for Material 1: Table II

BR	s1MFH	s1MFPhi	s1MFFl	s2MFH	s2MFPhi	s2MFFl	s3MFH	s3MFPhi	s3MFFl	s4MFH	s4MFPhi	s4MFFl	v1H	v1Phi	v1Fl	v2H	v2Phi	v2Fl	v3H	v3Phi	v3Fl	v4H	v4Phi	v4Fl
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The boundary condition flux for each variable Var can be written as: BCFlux = FaceArea*VarH*(VarPhi - Var) + FaceArea*VarFl

File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
BcDefinitions BcData
AllFluids Material 1 Material 2 Material 3 Material 4

Figure 34. GUI Boundary Conditions "BcData" sub-tab ("Material 1" sub-tab).

Section 4.7.1 describes the available boundary condition types and kinds. Section 4.7.2 gives a complete listing of the **Boundary Condition** panel input specifications.

4.7.1. Boundary Condition Types and Kinds

type: keyword text; either `external` or `internal`.
CartaBlanca can apply boundary conditions either at nodes which are on the border of the computational mesh, or are at interior locations. These basic types of boundary condition are specified with the keywords `external` and `internal` for the **type** parameter, respectively. Normally, a boundary is `external`, however, an `internal` boundary can be used for some special purposes. For example, if one wants to set velocity to zero for some internal nodes ("sticky nodes"), then one can put those nodes in a boundary section with type `internal` and **kind** `wall`, and set a large value (such as1.0E20) for **VxH**, **VyH**, **VzH**, and 0.0 for **VxPhi**, **VyPhi**, **VzPhi** (see the discussion on `wall` boundaries below).

kind: keyword text; either `wall`, `inflow`, `outflow`, `pressure`, `inflow-outflow`, `reflective`, `vel-direction`, or `reflcorner`.
CartaBlanca supports eight different boundary condition algorithms, which gives a great deal of flexibility for problem definition. A specific algorithm is chosen with the **kind** parameter keywords; the algorithms are listed in Table 1, after which detailed descriptions for each are given. All nodes on the border of the computational mesh ("external" nodes) that are not explicitly included in a boundary condition region by the problem input are included in a default boundary of **kind** `wall`. If there are no boundary regions at all in the input, then all external nodes are in this default `wall` boundary (with the exception of any nodes that treated by a periodic boundary condition (see Section 4.1.3).

Algorithm (kind-parameter keyword)	**Description**
`wall`	Specifies a constant boundary temperature (or heat flux) and velocity. By default, the wall is adiabatic and outward velocity component is 0.0. Can be used to hold materials fixed in space, including internal barriers. Default **kind**.
`inflow`	Specifies flows of mass, momentum, enthalpy, and species-mass into the problem domain.
`outflow`	Specifies flows of mass, momentum, enthalpy, and species-mass out of the problem domain.
`pressure`	Sets a constant pressure at the boundary. See discussion for special use of input parameter **ThetaH**.
`inflow-outflow`	Alternative method to set a constant pressure at the boundary. See discussion for special use of input parameter **ThetaH**.
`reflective`	Used in special cases for problem symmetry, such as the axis of the cylindrical coordinate system in the axisymmetric case.
`vel-direction`	For nodes in a `vel-direction` boundary, velocity, if any, is constrained to a specified direction. Used for special locations, such as corner nodes in a cylindrical coordinate system.
`reflcorner`	Allows both positive and negative component of a vector (velocity, pressure gradient, etc.) in a specified direction. Used for special locations.

Table 1. Boundary conditions.

Internally, the boundaries are ordered by the boundary region number, followed by the default `wall` boundary. If a node has been claimed by a boundary, then it is out of consideration by the boundaries that follow that boundary, that is, a node cannot belong to more than one boundary.

In general, if a flux condition in a boundary condition is applicable (as described below), then it is defined as

$$BCFlux = FaceArea * VarH * (VarPhi - Var) + FaceArea * VarFl \ .$$

For example, energy flux at a boundary is calculated in the code by using, with the variable Temp (temperature), the relation

$$BCFlux = FaceArea*TempH*(TempPhi - Temp) + FaceArea*TempFl \ .$$

<u>**kind** `wall`</u>

`wall` boundary conditions are used to specify a constant boundary temperature (or heat flux) and velocity. By default, a `wall` boundary is adiabatic (if energy transport is solved), the outward normal velocity is set to 0.0, and inward normal and tangential velocity are allowed. A `wall` boundary can be used to hold materials fixed in space, including the creation of internal barriers.

The `wall` boundary condition for temperature T is assumed to have the form

$$k\frac{\partial T}{\partial n} = -h(T - T_\infty) + q \quad ,$$

where *k* is the heat conductivity as used in the energy equation, *n* is the outward normal direction, *h* is the heat transfer coefficient, T_∞ is the ambient temperature, and *q* is the heat flux. The **BcData** panel's temperature parameters **TempH**, **TempPhi**, and **TempFl** correspond to *h*, T_∞, and *q* respectively in the temperature boundary condition differential equation.

Internally, the code currently sets a Dirichlet-type boundary condition flag (dependent variable's value is specified at the boundary) when **TempH** is greater than 10^9, and directly sets the boundary node temperatures to **TempPhi** (a very large value of the heat transfer coefficient would have the same effect using the differential equation). Alternatively, `wall` can be used to specify a boundary heat flux with the "Temp" parameters. By default a `wall` boundary is adiabatic.

<u>Example: set fixed-temperature boundaries</u>: If the energy transport equation is solved (**Physics** panel), to set the boundary temperatures at, say, fixed values T = 300 and T = 500 for boundary regions 1 and 2 for phases 1 and 3, in the **Material 1** and **Material 3** panels enter a large number (such as 1.0E20) in the first and second rows under **TempH**, enter 300.0 and 500.0 under **TempPhi**, and enter 0.0 under **TempFl**. This sets the boundary temperatures to the desired values.

<u>Example: set adiabatic boundary</u>: If the energy transport equation is solved (**Physics** panel), to set an adiabatic boundary for boundary region 5, for phase 4, in the **BcData** panel's **Material 4** panel set the 5th row's **TempH**, **TempPhi**, and **TempFl** to 0.0 (this is the default).

Example: set velocity at a boundary: Specify velocity on a wall by setting **VxH**, **VyH**, and **VzH** to a large value (such as 1.0E30), use **VxPhi**, **VyPhi**, and **VzPhi** to specify the desired velocity on the boundary, and set **VxFl**, **VyFl**, and **VzFl** to 0.0.

Example: default `wall` velocity behavior: If **VxH**, **VxPhi**, **VxFl**, **VyH**, **VyPhi**, **VyFl**, **VzH**, **VzPhi**, and **VzFl** are 0.0 (the default values) on a `wall` boundary, the treatment of velocity on the boundary nodes is to set the outward (in the normal direction of the boundary face) velocity to zero. The code will still calculate tangential velocity and inward velocity.

Most of the problems in the CartaBlanca Test Suite have explicit definition of at least one `wall` boundary condition.

`wall` boundary condition treatment for species transport and turbulence transport are currently not available.

`wall` is the default boundary condition **kind**. All external computational nodes that are not explicitly included in a boundary condition region of the problem geometry are included by the code in a `wall` boundary, using default settings.

kind `inflow`

The `inflow` boundary condition is used to specify flows (fluxes) of mass, momentum, enthalpy, and species-mass into the problem domain at boundary nodes, using "Phi" parameters from the **BcData** panel. The velocity parameters **VxPhi**, **VyPhi**, and **VzPhi** must be provided (depending of course on the problem's dimensionality) for each material (phase); they are used to calculate a volumetric flux into the problem domain through the face of a boundary node. The fluxed quantities for mass, momentum, enthalpy, and species-mass are calculated using the volumetric flux. The mass flux is calculated using, for each material (phase) **ThetaPhi** (volume fraction) and (if energy transport is being solved) **TempPhi** (temperature), and, from the **AllFluids** subpanel, **PPhi** (pressure). Momentum flux is calculated using the mass flux and **VxPhi**, **VyPhi**, and **VzPhi**. Enthalpy flux is calculated using the mass flux, **TempPhi**, and the species mass fraction parameters (again, for each material (phase)) **s1MFPhi** …. **s4MFPhi**. Species-mass flux is calculated using **s1MFPhi** …. **s4MFPhi**.

Examples: `inflow` boundary condition: The CartaBlanca Test Suite contains seven problems that use an `inflow` boundary condition; these are all coupled with a corresponding `pressure` boundary condition (see discussion below on the `pressure` boundary condition). For example, `testParticleTranslation.IO` is a 1-D two-phase (solid and air) problem that only solves for momentum transport. At the start of the problem the solid (modeled with particles) and air move to the left (-x direction) at the same velocity. There is an `external pressure` boundary condition at $x = 0$, set to the pressure in the problem domain, and an `external inflow` boundary condition at $x = 1.0$ that sends-in air at the initial air/solid problem velocity. The solid and air translate to the left at their initial velocity.

kind `outflow`

The `outflow` boundary condition is used to specify flows (fluxes) of mass, momentum, enthalpy, and species-mass out of the problem domain at boundary nodes. The user must provide values in the **BcData** panel for **VxPhi**, **VyPhi**, and **VzPhi**. All other quantities needed for the flux calculations are taken by the code from the current state of the `outflow` boundary node(s).

kind `pressure`

The `pressure` boundary condition is used to specify the pressure at a boundary. Internally, for numerical reasons, in addition to setting the specified pressure at the boundary node(s), the code must adjust material densities at `pressure` nodes by bringing in materials (inflow) or ejecting materials (outflow).

The user supplies the desired pressure for each boundary condition region with **kind** `pressure`, in the GUI **BcData** panel's **AllFluids** subpanel, with parameter **PPhi** (pressure). Also, for the code's inflow/outflow logic for `pressure` boundaries, the user supplies volume fraction information for each material (phase) with the **ThetaPhi** and **ThetaH** parameters. Parameter **ThetaPhi** specifies each material's volume fraction to use when the code detects that inflow is required. Parameter **ThetaH** is only used as a special flag to control the choice of volume fractions to use if outflow is required. If **ThetaH** is equal-to or greater-than 1.0E99 for at least one of the materials, the code will use the inflow **ThetaPhi** values also for the outflow case (for all of the materials). For a solid-gas problem, typically only the gas would be expected to leave the system, and the **ThetaH** flag can be used to enforce this. If **ThetaH** is less than 1.0E99 (for all of the materials), the code will use the boundary node's current volume fractions for the outflow condition. As a final step in the `pressure` logic, the code adjusts, for inflow conditions, the node-values of enthalpy, momentum, and species mass using user-supplied values of temperature (**TempPhi**), velocity (**VxPhi**, **VyPhi**, and **VzPhi**), and species mass fractions (**s1MFPhi** …. **s4MFPhi**).

Examples: `pressure` boundary condition: The CartaBlanca Test Suite contains eight problems that use a `pressure` boundary condition; seven of these are coupled with an `inflow` boundary condition (see discussion above on the `inflow` boundary condition).
For example, as described above for `inflow`, `testParticleTranslation.IO` is a 1-D two-phase (solid and air) problem that only solves for momentum transport. At one end of the problem domain there is an `external pressure` boundary condition, and at the other end an `external inflow` boundary condition (see additional discussion above for `inflow`). Problem `testParticleSinglePhaseTranslation.IO` is similar to `testParticleTranslation.IO`, but there is only a single (particle) phase, and a single `external pressure` boundary condition at $x = 0$, set to the pressure in the problem domain. The solid translates to the left at its initial velocity.

kind `inflow-outflow`

The `inflow-outflow` boundary condition is also used to specify the pressure at a boundary; it is an alternative way to implement a `pressure` boundary condition in special circumstances. In the code's implementation of the `inflow-outflow` boundary condition, the specified pressure is not set on the node directly, rather, it is used to determine boundary fluxes for adjusting material densities, and the equation of state is used to adjust the pressure. In this way, the resulting pressure may not be exactly the specified pressure.

The `inflow-outflow` input parameters are used in a manner that is analogous to the `pressure` boundary condition logic. The user supplies the desired pressure for each boundary condition region with **kind** `inflow-outflow`, in the GUI **BcData** panel's **AllFluids** subpanel, with parameter **PPhi** (pressure).

For the boundary-flux logic that is used to adjust the material densities, the user supplies volume fraction information for each material (phase) with the **ThetaPhi** and **ThetaH** parameters. Parameter **ThetaPhi** specifies each material's volume fraction to use when the code detects that inflow is required. The material density to use for the inflow flux density is calculated from the user-supplied boundary pressure, temperatures (**TempH** parameter, if energy transport is solved), and species mass fractions (**s1MFPhi** …. **s4MFPhi**). Parameter **ThetaH** is used as a special flag to determine the flux density if outflow is required. If **ThetaH** is equal-to or greater-than 1.0E99 for at least one of the materials, the code will calculate the outflow flux density as the (old-time) local density times the **ThetaPhi** value (for all of the materials). For a solid-gas problem, typically only the gas would be expected to leave the system, and the **ThetaH** flag can be used to enforce this. If **ThetaH** is less than 1.0E99 (for all of the materials), the code will use the boundary node's (old-time) local material densities for the outflow condition.

As a final step in the `inflow-outflow` logic, the code adjusts, for inflow conditions, the node-values of enthalpy, momentum, and species mass using user-supplied values of temperature (**TempPhi**), velocity (**VxPhi**, **VyPhi**, and **VzPhi**), and species mass fractions (**s1MFPhi** …. **s4MFPhi**).

> Usage note: Although the coding is intended for general cases, currently the `inflow-outflow` boundary condition has been used only with two phases, and for the outflow case using the **ThetaH** flag $\geq$ 1.0 x 10^{99}, with **ThetaPhi** = 1 or 0.

kind `reflective`

The `reflective` boundary is used in special cases for problem symmetry, such as the axis of the `cylindrical` coordinate system in the axisymmetric case. The normal component of the velocity is set to zero, as well as the normal directional derivative of the velocity components used in calculating viscous stress, and the normal directional derivative of the pressure. For energy transport, a `reflective` boundary is treated as adiabatic. See also discussions below on the `vel-direction` and `reflcorner` boundary conditions.

> Examples: `reflective` boundary condition: The CartaBlanca Test Suite contains six problems that use either one or two `reflective` boundary conditions; five of these problems are in `cylindrical` coordinates, and one is `cartesian`. For example, problem `testParticleCylindrical.IO` calculates the impact of a projectile onto a target in 2-D `cylindrical` coordinates, where *x* = 0 is a rotational symmetry axis. An `external reflective` boundary condition is specified for nodes at *x* = 0, *y* > 0 (the point *x* = y = 0 is treated as a special case with a `vel-direction` boundary condition, as described below). Note that `testParticleCylindrical.IO` also specifies a reflective boundary condition for *x* > 0, *y* = 0; the *x*-axis is not a symmetry axis, and a `wall` boundary could also be used here (the reflective boundary suppresses a wall-rebound).

kind `vel-direction`

The `vel-direction` boundary condition is used to restrict velocities to a direction that is specified with the **VxPhi**, **VyPhi**, and **VzPhi** parameters in the **BcData** panel for each material. (The code automatically normalizes the resultant direction vectors to have magnitude one.) Only velocity in the positive direction of the specified direction is allowed for `vel-direction` boundary nodes. This would be used, for example, at a corner node which is the intersection of the axisymmetric axis (in the *y* direction) and a wall (in the *x* direction) in the `cylindrical` coordinate system. For such a node, we should allow only positive *y*-velocity, and the `vel-direction` boundary condition can be used. Note that, for this boundary condition, the normal component of the pressure gradient and the normal directional derivative of the velocity components used in calculating viscous stress are not set to zero.

Example: `vel-direction` boundary condition: Test Suite problem `testParticleCylindrical.IO` is in 2-D `cylindrical` coordinates, where the *y*-axis is a rotational-symmetry axis (see discussion above on the `reflective` boundary condition). The point $x = \mathrm{y} = 0$ is treated with an `external` `vel-direction` boundary condition; for each phase (projectile, target, and air), in the **BcData** panel, **VxPhi** = 0.0 and **VyPhi** = 1.0. Note that $x = \mathrm{y} = 0$ is specified with boundary region 3; it is only necessary to specify boundary region 3 as $y = 0$, because region 2 has claimed $y = 0, x > 0$.

kind `reflcorner`

The `reflcorner` boundary condition is similar to the `vel-direction` boundary, but it allows both positive and negative components of a vector (velocity, pressure gradient, etc.) on the specified direction. As with `vel-direction`, this direction is given with the **VxPhi**, **VyPhi**, and **VzPhi** parameters for each material in the **BcData** panel. Only the component of pressure gradient and velocity gradient in the specified direction is allowed. The `reflcorner` boundary is used, for example, at the intersection line of two symmetry planes represented as `reflective` boundaries.

Example: `reflcorner` boundary condition in 3-D: In a 3-D case, if the planes $x = 0$, $y = 0$, and $z = 0$ are symmetry planes and we model an eighth part of a sphere $(x \geq 0, y \geq 0, z \geq 0)$, then the *yz*-plane ($x = 0, y > 0, z > 0$) is `reflective`, as are the *xz*- and *xy*-planes. The positive *x*-axis can be specified as `reflcorner`, with **VxPhi** = 1.0, **VyPhi** = **VzPhi** = 0.0; the positive *y*-axis and the positive *z*-axis can be specified as `reflcorner` similarly. The origin can be specified as `reflcorner` with **VxPhi** = **VyPhi** = **VzPhi** = 0.0. In the case of the origin, the velocity, pressure gradient, and velocity gradient will be set to zero. If not all **VxPhi**, **VyPhi**, and **VzPhi** are equal to zero, the code normalizes the direction so the magnitude of the direction is one.

4.7.2. Input Specifications

The input parameters for the **Boundary Conditions** panel are as follows:

BcDefinitions panel

numBCConics: integer; number of surfaces that will be used to specify the boundary condition regions.

numBCRegions: integer; number of boundary condition regions.

tolerance: real, default value = 1.0 x 10^{-8}; tolerance for accepting a point as part of a boundary condition region.

Note: The value of **tolerance** is also used for the **Initial Conditions** regions (Section 4.6).

SurType: keyword text; the surface type, either `Conic` or `FilledBox`. Entered for each surface in the surface table.

A, B, C, D, E, F, G, H, I, J: ten reals; one set of values for each surface. For **SurType** `Conic`, the coefficients of the 3-D conic surface
$h(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz + Gx + Hy + Iz + J$.
For **SurType** `FilledBox`, the opposite vertices (x_1, y_1, z_1) and (x_2, y_2, z_2) of a rectangular parallelepiped, where A = x_1, B = y_1, C = z_1, D = x_2, E = y_2, and F = z_2 (G, H, I, and J are not used). For `FilledBox` in a 2-D problem, C and F are not used and should be set to zero.

lt, le, eq, ge, gt: five text fields; one set of entries for each boundary condition region. For **SurType** `Conic`, the surface id numbers are entered to specify the relations $h(x,y,z) < 0.0$, $h(x,y,z) = 0.0$, or $h(x,y,z) > 0.0$. For **SurType** `FilledBox`, the surface id numbers are used to specify if a point is in the box, where "lt" has the sense of including the point, "gt" is used to exclude the point, etc. ("eq" should not be used). For `Conic` and `FilledBox` surfaces, where more than one surface is used for a relation, the id numbers must be separated by commas, with no embedded blanks; if a relation does not apply for a region (i.e., no surface satisfies the relation), "-1" should be entered. Surface types may be mixed in a given relation. If all five relations have "-1" for a region, a file that gives the region's nodes must be provided (see discussion above in this section, and Section 4.1).

type: keyword text; the boundary region type, either `internal` or `external`. See Section 4.7.1.

kind: keyword text; the boundary region kind, either `wall`, `reflective`, `reflcorner`, `inflow`, `outflow`, `pressure`, `inflow-outflow`, or `vel-direction`. See Section 4.7.1.

BcData panel

AllFluids panel

PH, PPhi, PFl: three reals; for each boundary condition region, pressure flux terms.

tkH, tkPhi, tkFl: three reals; for each boundary condition region, K-parameter (turbulent kinetic energy) flux terms for K-epsilon turbulence model.

tlH, tlPhi, tlFl: three reals; for each boundary condition region, ε-parameter (turbulent dissipation) flux terms for K-epsilon turbulence model.

Note: Currently, input cells for the tk and tl parameters are placeholders.

Material 1, **Material 2**, **Material 3**, and **Material 4** panels

ThetaH, ThetaPhi, ThetaFl: three reals; for each boundary condition region, volume fraction flux terms for Material 1, 2, 3, or 4.

TempH, TempPhi, TempFl: three reals; for each boundary condition region, temperature flux terms for Material 1, 2, 3, or 4.

VxH, VxPhi, VxFl: three reals; for each boundary condition region, X-component of velocity flux terms for Material 1, 2, 3, or 4.

VyH, VyPhi, VyFl: three reals; for each boundary condition region, Y-component of velocity flux terms for Material 1, 2, 3, or 4.

VzH, VzPhi, VzFl: three reals; for each boundary condition region, Z-component of velocity flux terms for Material 1, 2, 3, or 4.

DxH, DxPhi, DxFl: three reals; for each boundary condition region, X-component of displacement flux terms for Material 1, 2, 3, or 4.

DyH, DyPhi, DyFl: three reals; for each boundary condition region, Y-component of displacement flux terms for Material 1, 2, 3, or 4.

DzH, DzPhi, DzFl: three reals; for each boundary condition region, Z-component of displacement flux terms for Material 1, 2, 3, or 4.

s1MFH, s1MFPhi, s1MFFl: three reals; for each boundary condition region, species-1 mass fraction flux terms for Material 1, 2, 3, or 4.

s2MFH, s2MFPhi, s2MFFl: three reals; for each boundary condition region, species-2 mass fraction flux terms for Material 1, 2, 3, or 4.

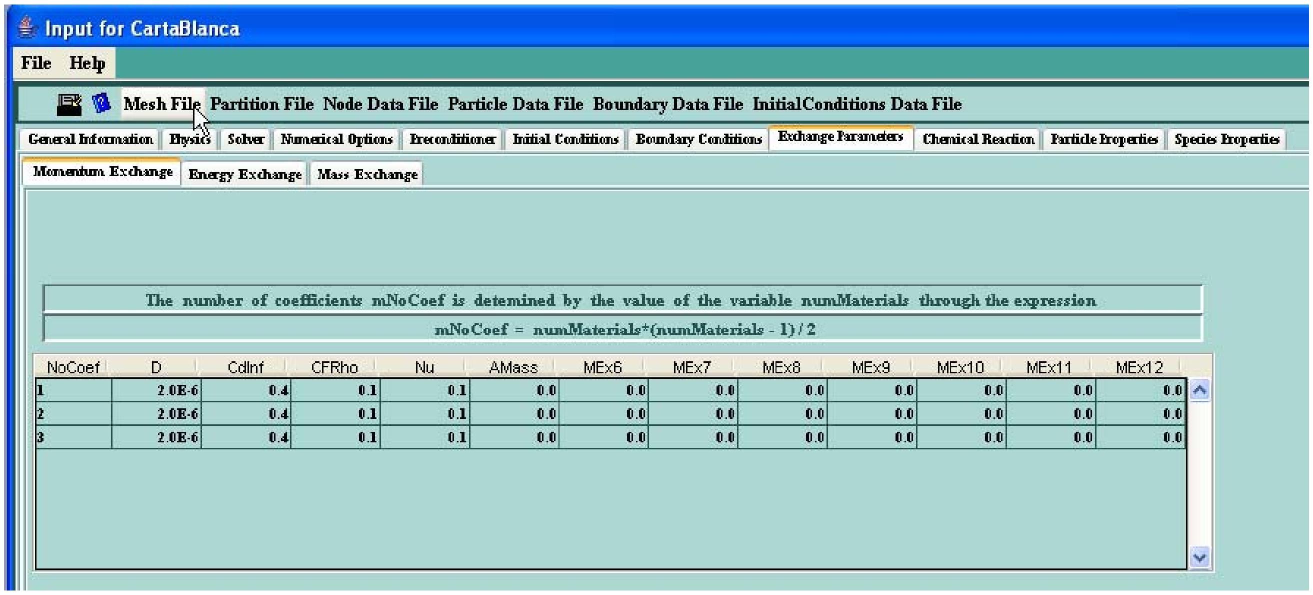
s3MFH, s3MFPhi, s3MFFl: three reals; for each boundary condition region, species-3 mass fraction flux terms for Material 1, 2, 3, or 4.

s4MFH, s4MFPhi, s4MFFl: three reals; for each boundary condition region, species-4 mass fraction flux terms for Material 1, 2, 3, or 4.

v1H, v1Phi, v1Fl, v2H, v2Phi, v2Fl, v3H, v3Phi, v3Fl, v4H, v4Phi, v4Fl:12 reals; currently not used.

4.8. Exchange Parameters

CartaBlanca can solve transport equations for up to four independent phases (each of which can consist of more than one species), which are typically identified as Material 1, 2, 3, and 4 in the GUI. Physical interactions between the phases are specified with the GUI **Exchange Parameters** panel, which has three sub-panels: **Momentum Exchange**, **Energy Exchange** and **Mass Exchange**; these are shown in Figures 35, 36, and 37, respectively.

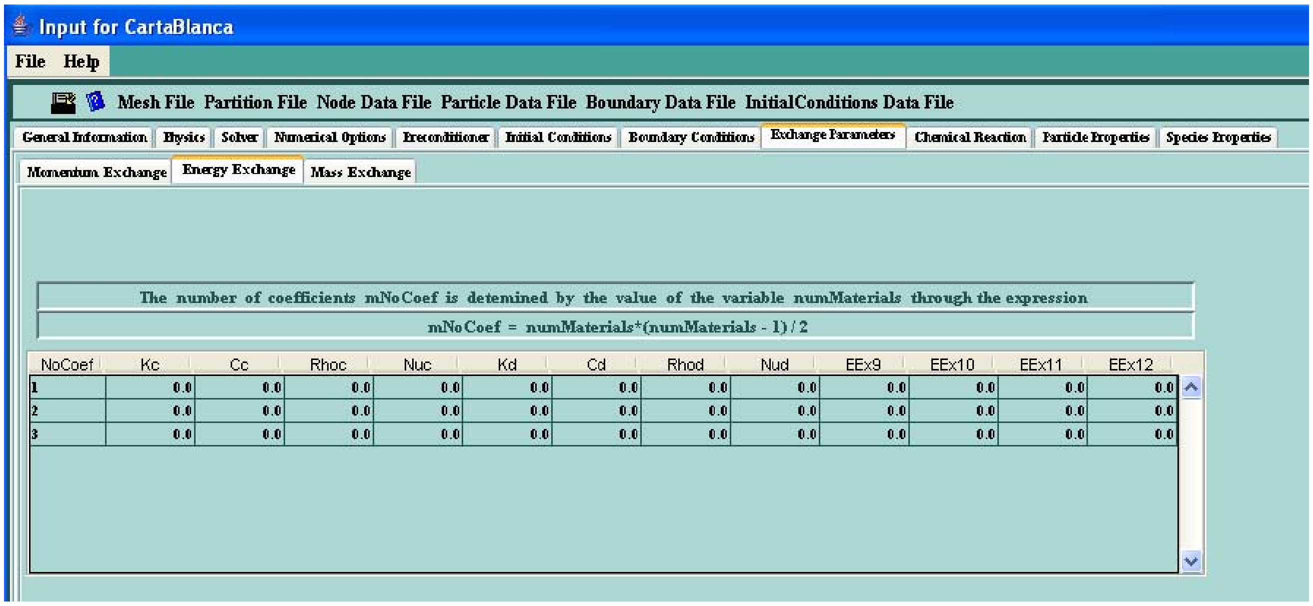

Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
Momentum Exchange Energy Exchange Mass Exchange

The number of coefficients mNoCoef is detemined by the value of the variable numMaterials through the expression

mNoCoef = numMaterials*(numMaterials - 1) / 2

NoCoef	D	CdInf	CFRho	Nu	AMass	MEx6	MEx7	MEx8	MEx9	MEx10	MEx11	MEx12
1	2.0E-6	0.4	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2.0E-6	0.4	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	2.0E-6	0.4	0.1	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 35. GUI "Exchange Parameters" tab ("Momentum Exchange" sub-tab).

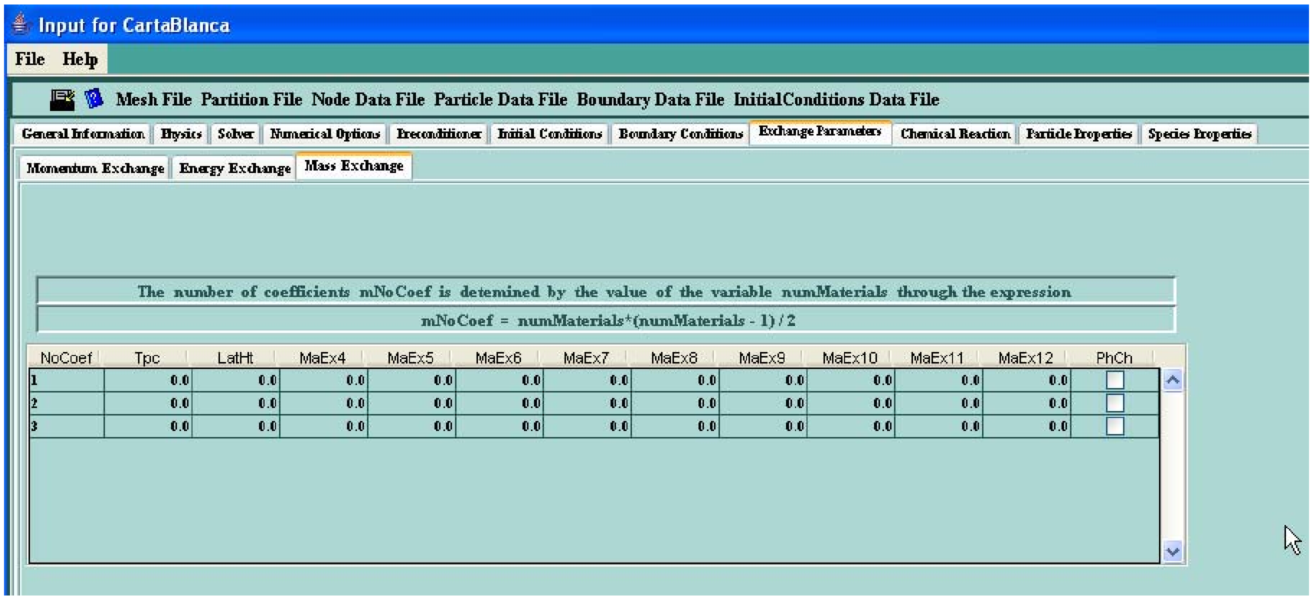

Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
Momentum Exchange Energy Exchange Mass Exchange

The number of coefficients mNoCoef is detemined by the value of the variable numMaterials through the expression

mNoCoef = numMaterials*(numMaterials - 1) / 2

NoCoef	Kc	Cc	Rhoc	Nuc	Kd	Cd	Rhod	Nud	EEx9	EEx10	EEx11	EEx12
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 36. GUI "Exchange Parameters" tab ("Energy Exchange" sub-tab).


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information Physics Solver Numerical Options Preconditioner Initial Conditions Boundary Conditions Exchange Parameters Chemical Reaction Particle Properties Species Properties
Momentum Exchange Energy Exchange Mass Exchange

The number of coefficients mNoCoef is determined by the value of the variable numMaterials through the expression

mNoCoef = numMaterials*(numMaterials - 1) / 2

NoCoef	Tpc	LatHt	MaEx4	MaEx5	MaEx6	MaEx7	MaEx8	MaEx9	MaEx10	MaEx11	MaEx12	PhCh
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	☐
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	☐
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	☐

Figure 37. GUI "Exchange Parameters" tab ("Mass Exchange" sub-tab).

Each of these panels contains a table, where each row is used to specify various interactions (exchange coefficients) between two of the problem's phases; the number of rows is automatically set by the code according to the number of materials (phases) in the problem, allowing for all possible combinations of the phases. In general, the number of rows (or phase pairs) is

$$numberOfRows = numMaterials * (numMaterials - 1)/2 \;,$$

where *numMaterials* is the total number of phases (**numNonParticleMaterials** + **numParticleMaterials**, as specified with the **Physics** panel (Section 4.2)). For problems with 1, 2, 3, and 4 materials, each of the three tables has 0, 1, 3, and 6 rows, respectively.

The rows in each table are labeled by their indices in the first column (heading **NoCoef**), starting at 1. Each row is used to enter exchange parameters between phase *i* and phase *j*, where the relations between row number and *i*, *j* are listed here in Table 2 (*i* and *j* both start at 1, corresponding to Material 1, 2, 3, and 4).

CartaBlanca's phase interaction models are discussed in Chapter 3 of the Theory Manual. The input parameters for momentum exchange are applied to the equation for the force $\boldsymbol{f}_{ik}$ between phases *i* and *k*:

$$\boldsymbol{f}_{ik} = \theta_i \theta_k K_{ik} \rho_{ik}^0 \left(\tilde{\boldsymbol{u}}_{\boldsymbol{k}} - \tilde{\boldsymbol{u}}_i \right) + A_{ik} \rho_{ik}^0 \left(\frac{\partial \tilde{\boldsymbol{u}}_k}{\partial t} + \tilde{\boldsymbol{u}}_k \nabla \cdot \tilde{\boldsymbol{u}}_{\boldsymbol{k}} - \frac{\partial \tilde{\boldsymbol{u}}_i}{\partial t} - \tilde{\boldsymbol{u}}_i \nabla \cdot \tilde{\boldsymbol{u}}_i \right) \qquad ,$$

where θ_i and θ_k are the volume fractions of phases *i* and *k*, $K_{ik} = K_{ki}$ is the momentum exchange coefficient, $A_{ik} = A_{ki}$ is the added mass coefficient, ρ_{ik}^0 is a reference density specified in the user input, and $\tilde{\boldsymbol{u}}_{\boldsymbol{i}}$ and $\tilde{\boldsymbol{u}}_{\boldsymbol{k}}$ are average velocities of phases *i* and *k*. Currently the added mass coefficient A_{ik} is an input parameter and the momentum exchange coefficient is calculated as

Two Phase Problem		
Row (NoCoef)	**phase *i***	**phase *j***
1	1	2
Three Phase Problem		
Row (NoCoef)	**phase *i***	**phase *j***
1	1	2
2	1	3
3	2	3
Four Phase Problem		
Row (NoCoef)	**phase *i***	**phase *j***
1	1	2
2	1	3
3	1	4
4	2	3
5	2	4
6	3	4

Table 2. Phase pairs for exchange coefficient rows.

$$K_{ik} = \frac{3}{4} C_d \frac{\left|\tilde{\boldsymbol{u}}_{\boldsymbol{k}} - \tilde{\boldsymbol{u}}_{\boldsymbol{i}}\right|}{d_{ik}} \quad , \quad C_d = C_{d\infty} + \frac{24}{Re_{ik}} + \frac{6}{1+\sqrt{Re_{ik}}} \quad , \quad Re_{ik} = \frac{\left|\tilde{\boldsymbol{u}}_{\boldsymbol{k}} - \tilde{\boldsymbol{u}}_{\boldsymbol{i}}\right| d_{ik}}{\upsilon_{ik}} \quad ,$$

where the drag coefficient $C_{d\infty}$ for infinite Reynolds number, the length scale d_{ik}, and the kinematic viscosity υ_{ik} are input parameters.

In the following discussion on energy exchange, subscripts *i* and *j* again refer to phases *i* and *j*.

Currently, energy exchange is restricted in applicability; the basic model is derived from the literature for fluidized catalytic-cracking, where a thermal coupling (energy exchange) coefficient K_e is calculated as:

$$K_e = \frac{6 k_{ik} Nu}{d_{ik} d_{ik}} \quad ,$$

where k_{ik} is a user-input thermal conductivity, *Nu* a Nusselt number, and d_{ik} a user-input length scale. The Nusselt number is calculated as:

$$Nu = 2 + 0.6 \sqrt{Re_p} Pr^{1/3} \quad ,$$

where *Pr* is a Prandtl number and

$$Re_p = slip\left(\frac{d_{ik}}{\upsilon_{ik}}\right) \quad ,$$

where *slip* is the phasic slip speed (magnitude of the relative velocity), and υ_{ik} is a user-input kinematic viscosity. The Prandtl number is:

$$Pr = \frac{\upsilon_{ik}\rho_{ik}C_{ik}}{k_{ik}} \quad ,$$

where ρ_{ik} and C_{ik} are user-input density and specific heat (constant pressure), respectively.

There is also a special model for phasic heat exchange that was derived for study of the ignition of solid explosives. This model is invoked by selection of special energy and species systems in the **Physics Panel**: either `NLEnergyHE1` or `NLEnergyHE2`, and `NLSpeciesHE1`. In this case the solid-gas heat exchange is derived from the fluidized bed literature; the thermal coupling (energy exchange) coefficient K_e is calculated as:

$$K_e = \frac{6k_{ik}Nu}{d_{ik}d_{ik}\theta_g} \quad ,$$

where k_{ik} and d_{ik} are as defined above, θ_g is the gas volume fraction, and *Nu* is a Nusselt number that is calculated as:

$$Nu = 2 + 0.03Re_p^{1.3} \qquad (Nu \geq 2),$$

where Re_p is as calculated above.
Currently the modeling of phase change (mass exchange) is limited to several experimental phase change models implemented for chemical reactions related to high explosive materials.

The input parameters for the **Exchange Parameters** panel are as follows:

Momentum Exchange panel

D: real; length scale d_{ik}, a characteristic particle diameter.

Note: Parameter **D** is also used in the currently available energy exchange models.

CdInf: real; drag coefficient $C_{d\infty}$ for infinite Reynolds number.

CFRho: real; reference density ρ_{ik}^0. If there is at least one continuous fluid phase, it is recommended that the density of one of those phases be chosen for **CFRho**.

Nu: real; kinematic viscosity υ_{ik}.

AMass: real; added mass coefficient A_{ik}.

MEx6, MEx7, MEx8, MEx9, MEx10, MEx11, MEx12: seven reals; currently not used.

Energy Exchange panel

Note: The energy exchange parameters K, C, Rho, and Nu have either suffix “c” or suffix “d”, where “c” refers to a continuous phase and d to a disperse phase.

Note: Parameter **D** in the **Momentum Exchange** panel is also used by the energy exchange models.

Kc: real; thermal conductivity k_{ik}, continuous phase.

Cc: real; specific heat at constant pressure C_{ik}, continuous phase.

Rhoc: real; density ρ_{ik}, continuous phase.

Nuc: real; kinematic viscosity υ_{ik}, continuous phase.

Kd: real; thermal conductivity, disperse phase. Currently not used.

Cd: real; specific heat, disperse phase. Currently not used.

Rhod: real; density, disperse phase. Currently not used.

Nud: kinematic viscosity, disperse phase. Currently not used.

EEx9, EEx10, EEx11, EEx12: four reals; currently not used.

Mass Exchange panel

Tpc: real; phase change temperature.

LatHt: real; latent heat.

PhCh: boolean; true if there is a change of phase.

MaEx4, MaEx5, MaEx6, MaEx7, MaEx8, MaEx9, MaEx10, MaEx11, MaEx12: nine reals; currently not used.

4.9. Chemical Reaction

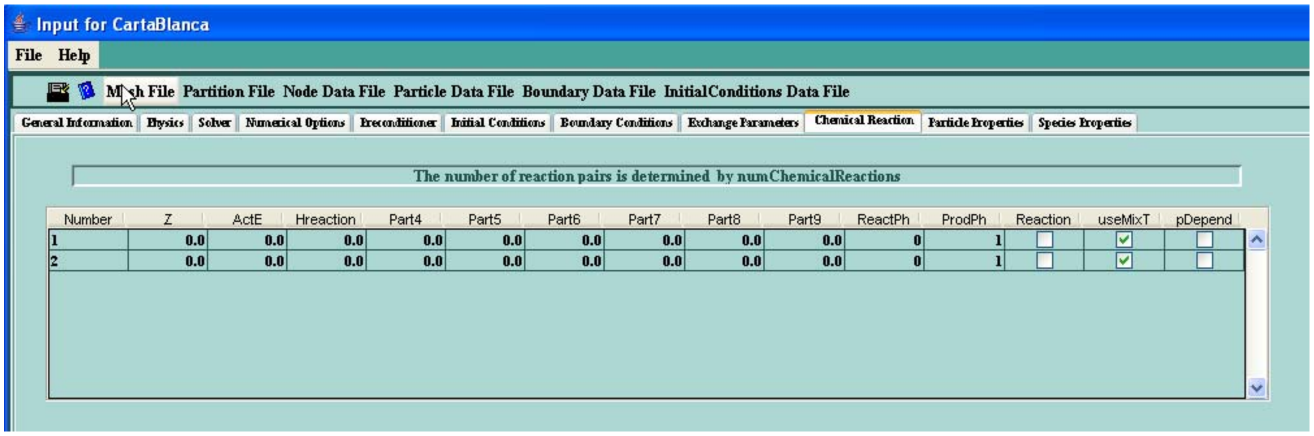
Chemical reactions can be modeled either with a rate constant equal to the Arrhenius term with a mixture temperature, or with a specialized gas-enhanced reaction model that was developed for study of solid explosive ignition. The Arrhenius term is

$$Z \exp(-E/RT) \,,$$

where Z is the frequency factor (pre-exponential factor, or steric factor), E is the activation energy, R is the universal gas constant, and T is the temperature.

Modeling of chemical reactions is enabled on the GUI **Physics** panel with the boolean **chemicalReactionOn**; if this is true, the number of chemical reactions to model in the problem is specified on the **Physics** panel with **numChemicalReactions** (see Section 4.2).

Data for the reactions are entered on the **Chemical Reaction** panel, which is shown in Figure 38. All data are entered in a single table, where each row is used for a reaction, and the rows are created automatically by the code according to the value of **numChemicalReactions** on the **Physics** panel.



The number of reaction pairs is determined by numChemicalReactions

Number	Z	ActE	Hreaction	Part4	Part5	Part6	Part7	Part8	Part9	ReactPh	ProdPh	Reaction	useMixT	pDepend
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	☐	✓	☐
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	1	☐	✓	☐

Figure 38. GUI "Chemical Reaction" tab.

The input parameters for the **Chemical Reaction** panel are as follows:

<u>Note</u>: Currently chemical reactions have only been modeled in CartaBlanca for cases using special species systems `NLSpeciesHE1` and `NLSpeciesHE2`, which are selected in the **Physics** panel. `NLSpeciesHE1` can only treat one reaction; `NLSpeciesHE2` treats two reactions with one reaction between gas species. The "`HE`" in these systems stands for high explosive. The basic Arrhenius reaction rate is available for these systems, but the actual high explosive modeling has been done with extensions to the simple Arrhenius term to model the explosive's thin combustion wave front; these extensions are invoked with the parameters **useMixT** and **pDepend**, as indicated below.

<u>Z</u>: real; frequency factor (pre-exponential factor, or steric factor) in Arrhenius reaction term.

<u>ActE</u>: real; activation energy in Arrhenius reaction term.

Note: In the Arrhenius reaction expression the units of the activation energy and the gas constant R must be consistent. R is specified in the **Physics** panel (Section 4.2); the default value is 8.31439 J/mole-K.

Hreaction: real; heat of reaction. Currently, **Hreaction** is not used; **Species Properties**-panel parameter **HForm** (Section 4.11) is used for heat release.

ReactPh: integer; index of the reaction phase, starting at 0 (e.g., for Material 1, **ReactPh** = 0).

ProdPh: integer; index of the product phase, starting at 0, indicating a reaction in which the phase **ReactPh** goes into **ProdPh**.

useMixT: boolean; if checked, use the Arrhenius term with a mixture temperature; if false, use the gas-enhanced chemical reaction model for high explosive modeling.

Reaction: boolean; currently not used.

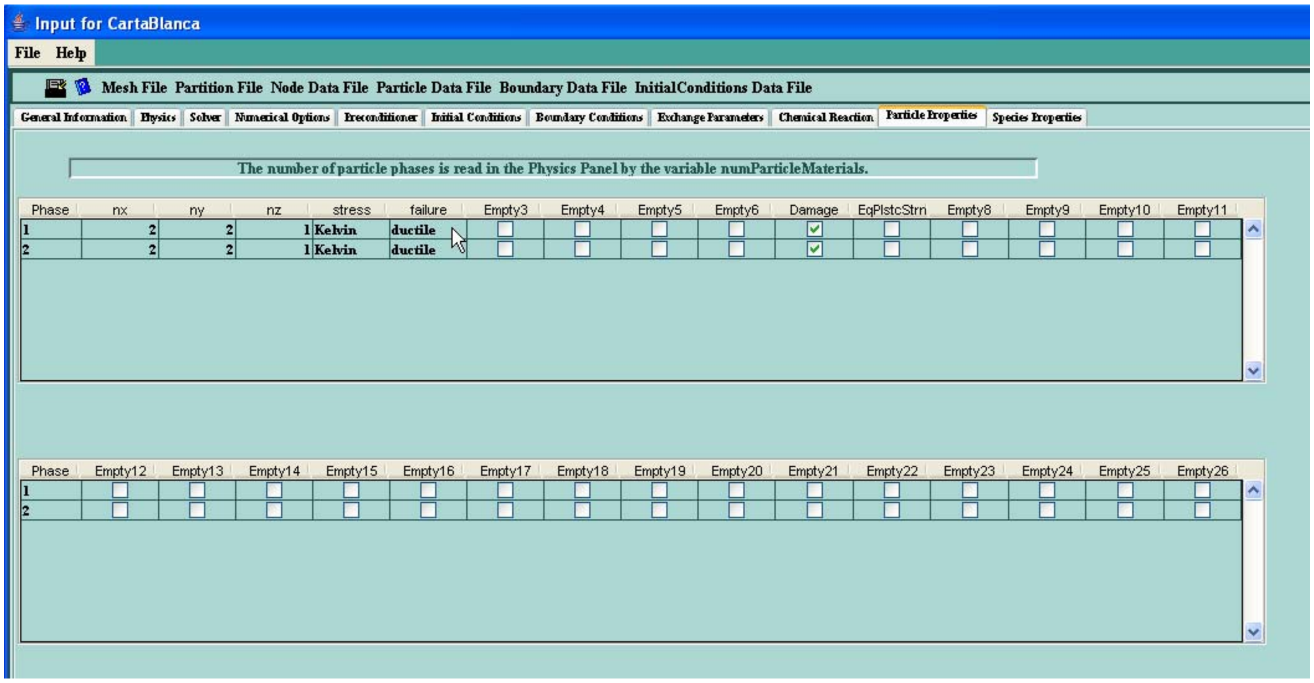
pDepend: boolean; if checked, use a pressure-dependent reaction model. This is part of the high explosive modeling logic, and is only considered if **useMixT** is false.

Part4, Part5, Part6, Part7, Part8, Part9: six reals; currently not used.

4.10. Particle Properties

The **Particle Properties** panel, which is shown in Figure 39, is used for data for phases modeled with the PIC/MPM method. There are two tables, each having up to four rows, where each row is used for one of the problem's PIC/MPM phases. The rows are created and numbered automatically, according to the value of **numParticleMaterials** in the **Physics** panel (Section 4.2).

The row-numbering requires some extra explanation. The first column of each table is labeled "Phase"; these values start at 1, but do <u>not</u> correspond to Material 1, 2, 3, 4 in the problem, but rather to the nth <u>particle</u> phase. For example, in a problem with Materials 1, 2, 3, and 4 being air (non-particle), steel (particle), water (non-particle), and aluminum (particle) respectively, the steel would be in the row labeled 1, and the aluminum in row 2.



The number of particle phases is read in the Physics Panel by the variable numParticleMaterials.

Phase	nx	ny	nz	stress	failure	Empty3	Empty4	Empty5	Empty6	Damage	EqPlstcStrn	Empty8	Empty9	Empty10	Empty11
1	2	2	1	Kelvin	ductile	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐
2	2	2	1	Kelvin	ductile	☐	☐	☐	☐	✓	☐	☐	☐	☐	☐

Phase	Empty12	Empty13	Empty14	Empty15	Empty16	Empty17	Empty18	Empty19	Empty20	Empty21	Empty22	Empty23	Empty24	Empty25	Empty26
1	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
2	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐

Figure 39. GUI "Particle Properties" tab.

Currently most of the cells in the tables are unused placeholders. The user must specify the number of computational particles per element (e.g., 2-D quad, 3-D hex) for the X, Y, and Z coordinates, and the use of one of the code's damage models. Note that additional material properties, including the damage model, are specified in the **Species Properties** panel (Section 4.11).

The input parameters for the **Particle Properties** panel are as follows:

<u>nx</u>: integer; number of computational particles per element for X-coordinate, for 1-D, 2-D, and 3-D problems.

<u>ny</u>: integer; number of computational particles per element for Y-coordinate, for 2-D, and 3-D problems.

<u>nz</u>: integer; number of computational particles per element for Z-coordinate, for 3-D problems.

<u>Damage</u>: boolean; if true, use one of the code's damage models (see **Species Properties** (Section 4.11)).

stress: keyword text; currently not used (see **Species Properties** (Section 4.11)).

failure: keyword text; currently not used (see **Species Properties** (Section 4.11)).

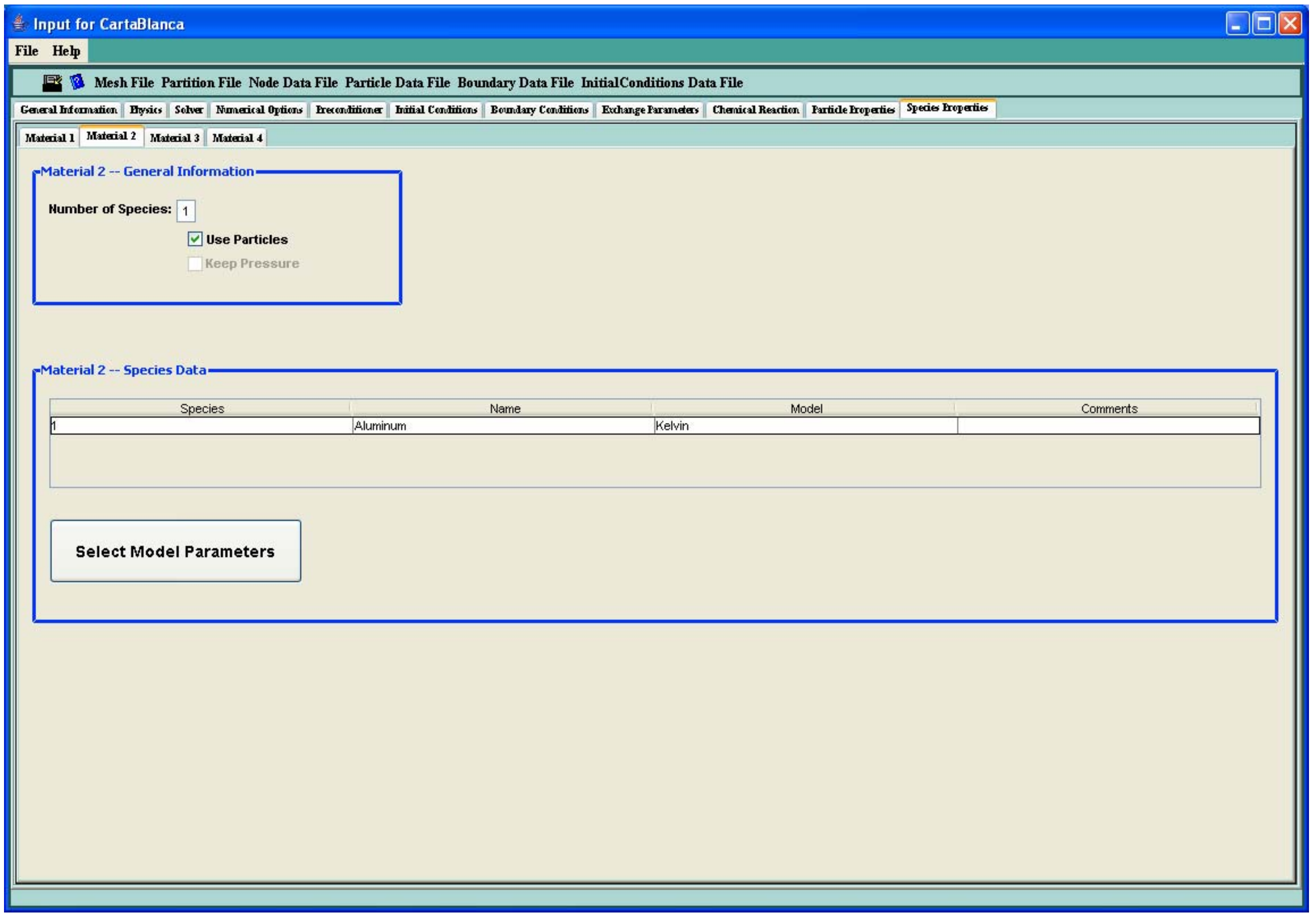
Empty3, Empty4, Empty5, Empty6: four booleans; currently not used.

EqPlstcStrn: boolean; currently not used.

Empty8,...…., Empty26: 19 booleans; currently not used.

4.11. Species Properties

Constitutive material properties, including equation-of-state and solid-stress modeling and data, are specified with the GUI **Species Properties** panel, which is shown in Figure 40.


Input for CartaBlanca
File Help
Mesh File Partition File Node Data File Particle Data File Boundary Data File InitialConditions Data File
General Information | Physics | Solver | Numerical Options | Preconditioner | Initial Conditions | Boundary Conditions | Exchange Parameters | Chemical Reaction | Particle Properties | Species Properties
Material 1 | Material 2 | Material 3 | Material 4
Material 2 -- General Information
Material 2 -- Species Data
Select Model Parameters

Number of Species: 1

[✓] Use Particles

[ ] Keep Pressure

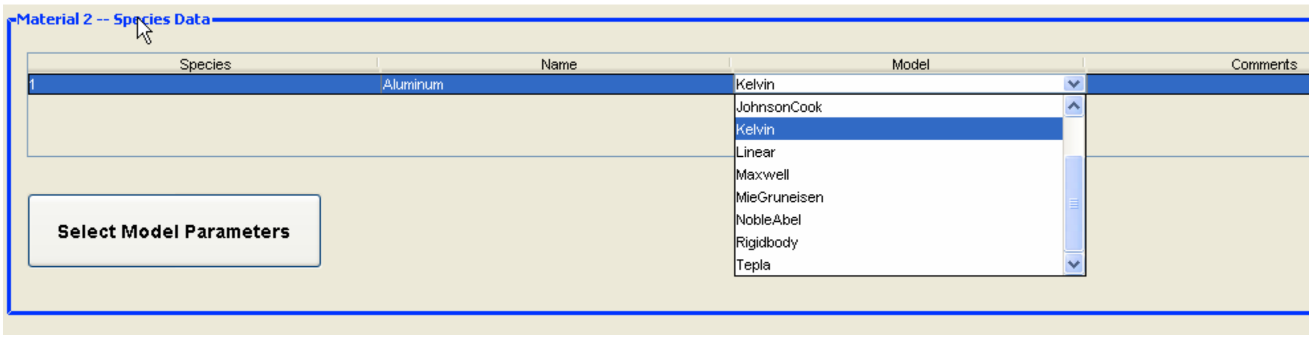
Species	Name	Model	Comments
1	Aluminum	Kelvin	

Figure 40. GUI "Species Properties" tab ("Material 2" sub-tab).

Species Properties comprises four sub-panels: **Material 1**, **Material 2**, **Material 3**, and **Material 4**. Each material panel has two areas for entering data: **General Information** and **Species Data**.

General Information is used for data that are global for that material (phase): **Number of Species** and the **Use Particles** checkbox (the variable Keep Pressure is currently unused). All species in a given phase have the same velocity and temperature fields, but each species may have its own constitutive model, and be specified individually in the initial and boundary conditions. The CartaBlanca Theory Manual describes the logic that treats a phase as a composite of its species' constitutive models. In principle there is no limit on the number of species that may be included in a phase, although the GUI currently does have an upper limit of four in the initial and boundary conditions panels. If **Use Particles** is checked, the MPM/PIC method is used for the phase; otherwise, the ALE method is used.

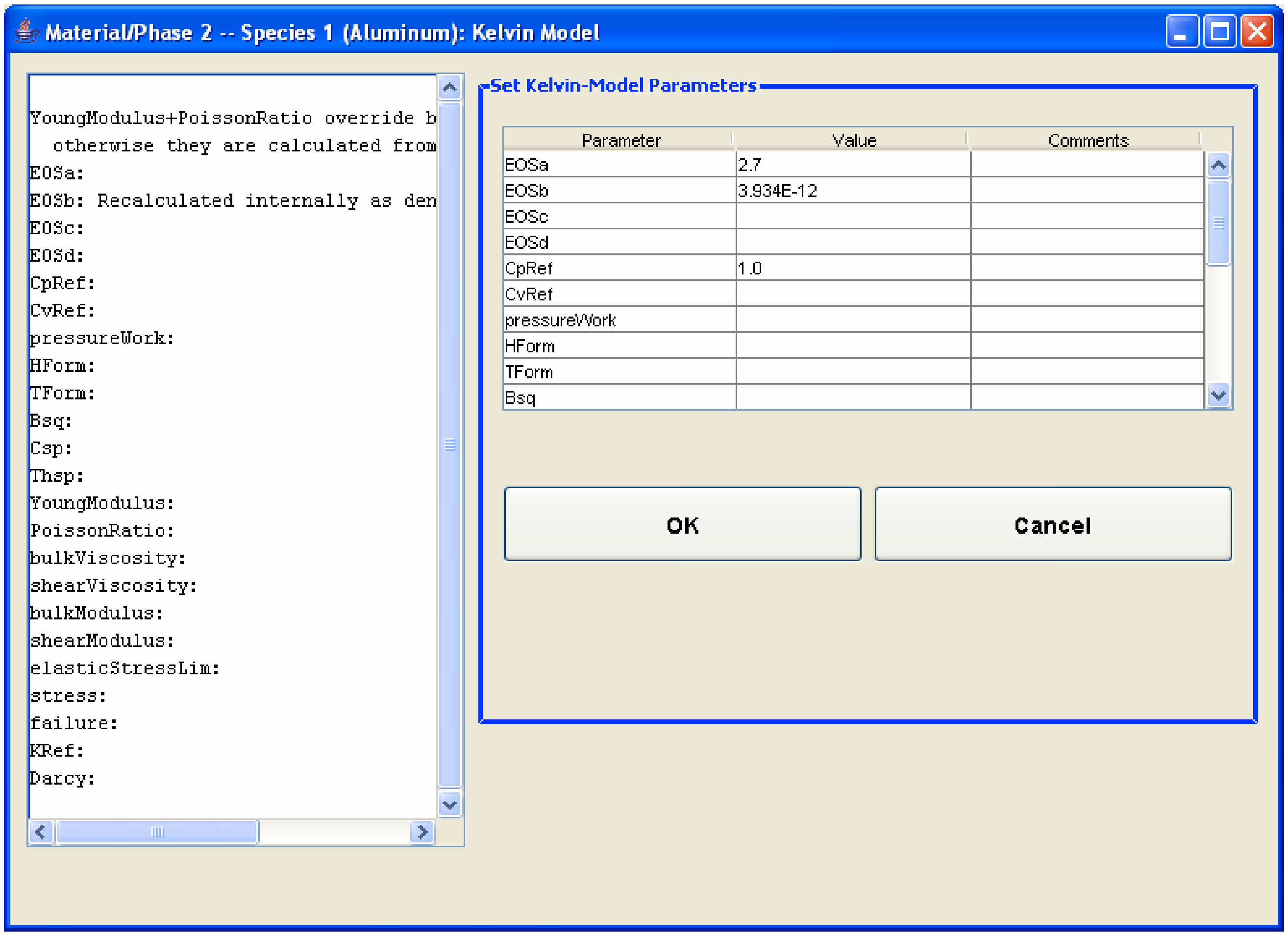
Species Data has a table that has a row for each species in the material (phase); the user can enter a name for the species and select a constitutive model from a built-in dropdown list (see Figure 41). Currently, the **Comments** table cells are not operational.


Material 2 -- Species Data

Species	Name	Model	Comments
1	Aluminum	Kelvin	

Figure 41. Selecting a species model.

With a model selected (and the species-row selected), the **Species Data -- Select Model Parameters** button (Figure 41) can be clicked to bring up a window for model-data entry (Figure 42).


Material/Phase 2 -- Species 1 (Aluminum): Kelvin Model

```
YoungModulus+PoissonRatio override b
  otherwise they are calculated from
EOSa:
EOSb: Recalculated internally as den
EOSc:
EOSd:
CpRef:
CvRef:
pressureWork:
HForm:
TForm:
Bsq:
Csp:
Thsp:
YoungModulus:
PoissonRatio:
bulkViscosity:
shearViscosity:
bulkModulus:
shearModulus:
elasticStressLim:
stress:
failure:
KRef:
Darcy:
```

Set Kelvin-Model Parameters

Parameter	Value	Comments
EOSa	2.7	
EOSb	3.934E-12	
EOSc		
EOSd		
CpRef	1.0	
CvRef		
pressureWork		
HForm		
TForm		
Bsq		

Figure 42. Entering data for a species model.

The data-entry window has a table on its right side, where each row is used to enter a value for a parameter used by the desired constitutive model (the Comments cells are currently not operational). Section 4.11.1

gives the top level of the **Species Properties** input specifications. Descriptions of the available material constitutive models, and their specific input specifications, are given in Section 4.11.2.

4.11.1. Input Specifications – Top Level

The input parameters for the **Species Properties** panel, for the **Material 1**, **2**, **3**, and **4** panels, are as follows:

General Information

Number of Species: integer; number of species included in the material (phase).

Use Particles: boolean; if true, use MPM/PIC method for phase. If false, use ALE.

Keep Pressure: boolean; currently not used.

Species Data – for each species in the phase

Name: text; any species identifying name user wishes to enter. No embedded blanks.

Model: keyword text; constitutive model for the species. See Table 3.

Comments: text; user-supplied comments. Currently not operational.

Select Model Parameters button

The **Select Model Parameters** button brings up a frame for data entry for the specific model selected. Section 4.11.2 contains complete lists of the parameters used by all of the available models.

4.11.2. Constitutive Models and their Input Specifications

The material constitutive models available in CartaBlanca are listed in Table 3, after which a more detailed description of each model, including its input specifications, is given.

Model	Description
`Rigidbody`	Solid material with constant density and no deformation.
`Incompressible`	Constant density.
`Linear`	Density linear in pressure (also 1/temperature term when energy equation solved). Used for fluids only.
`NobleAbel`	Used for gas.
`MieGruneisen`	Can be used for condensed matter and for materials under shock.
`Maxwell`	Constitutive model for solid stress in a Maxwell-type viscoelastic material.
`Kelvin`	Constitutive model for solid stress in a Kelvin-Voigt-type viscoelastic material [14].
`JohnsonCook`	Adds plasticity model to the Kelvin (Kelvin-Voigt) model [4].
`Tepla`	"Tensile Plasticity" model: plastic deformation with porosity growth [1, 2, 3].
`Sesame`	The Los Alamos SESAME Equation-of-State and Materials Properties Library [10].
`FortranModelOne`	Placeholder for user-provided Fortran model.
`GammaGas`	Only used when energy transport is solved; uses ratio of specific heats.
`ViscousSolid`	Experimental model used for code testing.

Table 3. Material models.

`Rigidbody`

This is a special constitutive relation; it is used when there is only one species in the phase and the deformation of the phase is negligible. In this constitutive relation the density of the material is a constant specified by user input in the data-entry window:

$$\rho^0 = \text{constant} \quad ,$$

where

ρ^0 = material density, and

constant = **EOSa**.

Also, the sound speed is set to zero so that the time step will not be affected by this phase.

The `Rigidbody` input parameters are as follows:

EOSa: real; material density (constant).

EOSb: real; not used.

EOSc: real; not used.

EOSd: real; not used.

CpRef: real; see `Maxwell` parameters.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

`Incompressible`

For an incompressible material the density is set to be a constant specified by user input in the data-entry window:

$$\rho^0 = \text{constant} \quad ,$$

where

ρ^0 = material density, and

constant = **EOSa**.

Also, the square of the sound speed is set to machine infinity (10^{64}).

The `Incompressible` input parameters are as follows:

EOSa: real; material density (constant).

EOSb: real; not used.

EOSc: real; not used.

EOSd: real; not used.

CpRef: real; see `Maxwell` parameters.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

shearViscosity: real; see `Kelvin` parameters.

KRef: real; see `Maxwell` parameters.

Darcy: real; see `Maxwell` parameters.

`Linear`

This is a constitutive model for a fluid with the following equation of state:

$$\rho^0 = \frac{A + Bp}{1 + C(T - D)} \quad ,$$

where

ρ^0 = material density,

A, *B*, *C* and *D* are model parameters specified by user input in the data-entry window,

A	=	**EOSa**,
B	=	**EOSb**,
C	=	**EOSc**,
D	=	**EOSd**,

and

p = pressure, and
T = temperature.

When energy transport is not solved (see Section 4.2), the equation of state is calculated as

$$\rho^0 = A + Bp \ .$$

The `Linear` input parameters are as follows:

EOSa: real; A-term in equation of state.

EOSb: real; B-term in equation of state.

EOSc: real; C-term in equation of state, only used when energy transport is solved.

EOSd: real; D-term in equation of state, only used when energy transport is solved.

CpRef: real; see `Maxwell` parameters.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

shearViscosity: real; see `Kelvin` parameters.

KRef: real; see `Maxwell` parameters.

Darcy: real; see `Maxwell` parameters.

`NobleAbel` gas

This is a constitutive model for a gas with the following equation of state:

$$\rho^0 = \frac{p}{Ap + BT} \quad ,$$

where

ρ^0 = material density,

A and B are model parameters specified by user input in the data-entry window,

A	=	**EOSa**,
B	=	**EOSb**,

and

p = pressure, and
T = temperature.

When energy transport is not solved (see Section 4.2), the equation of state is calculated as

$$\rho^0 = \frac{p}{Ap + B} \quad .$$

The `NobleAbel` input parameters are as follows:

EOSa: real; A-term in equation of state.

EOSb: real; B-term in equation of state.

EOSc: real; not used.

EOSd: real; not used.

CpRef: real; see `Maxwell` parameters.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

ReferenceTemperature: real; initial system temperature when not solving energy transport.

`MieGruneisen` equation of state

Mie-Gruneisen equations are often used for condensed matter and for materials under shock and impact. In this equation of state the density, enthalpy and pressure are related as

$$p = \frac{1}{1+\gamma}\left\{ p_h \left[1 - \frac{\gamma}{2}\left(\frac{\rho^0}{A} - 1 \right) \right] + \gamma \rho^0 \left(h - h_0 \right) \right\} \quad ,$$

where

$$p_h = \begin{cases} K_1\left(\frac{\rho^0}{A} - 1\right) & \text{if } \frac{\rho^0}{A} < 1 \\ K_1\left(\frac{\rho^0}{A} - 1\right) + K_2\left(\frac{\rho^0}{A} - 1\right)^2 + K_3\left(\frac{\rho^0}{A} - 1\right)^3 & \text{otherwise} \end{cases} \quad ,$$

γ, A, h_0, K_1, K_2 and K_3 are model parameters specified by user input in the data-entry window,

γ	=	**gamma**,
A	=	**EOSa**,
h_0	=	**EOSb**,
K_1	=	**K1**,
K_2	=	**K2**, and
K_3	=	**K3**.

and

p	=	pressure,
ρ^0	=	density, and
h	=	enthalpy.

The `MieGruneisen` input parameters are as follows:

EOSa: real; A-term in equation of state.

EOSb: real; reference enthalpy (h_0) in equation of state.

EOSc: real; not used.

EOSd: real; not used.

gamma: real; γ in equation of state.

K1: real; K_1 term in equation of state.

K2: real; K_2 term in equation of state.

K3: real; K_3 term in equation of state.

CpRef: real; see `Maxwell` parameters.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

shearViscosity: real; see `Kelvin` parameters.

KRef: real; see `Maxwell` parameters.

Darcy: real; see `Maxwell` parameters.

ReferenceTemperature: real; see `NobleAbel` parameters.

`Maxwell`

Viscoelastic materials can be modeled in CartaBlanca with either a Maxwell or Kelvin-Voigt model. Maxwell materials can be considered as a viscous damper (dashpot) in series with an elastic spring. The stress $\boldsymbol{\sigma}$ in this model is decomposed into pressure p and a deviatoric part $\boldsymbol{s}$ as

$$\boldsymbol{\sigma} = -p\boldsymbol{I} + \boldsymbol{s}.$$

The pressure is calculated as

$$\frac{dp}{dt} = -\frac{p}{\tau_p} + B\,tr(\dot{\boldsymbol{\varepsilon}}) + \gamma \mathbf{s} : \ddot{\boldsymbol{\varepsilon}}_d \quad ,$$

where τ_p is the relaxation time for pressure, B is the bulk modulus, $\dot{\boldsymbol{\varepsilon}}$ is the strain rate, γ is the Gruneisen coefficient, and $\ddot{\boldsymbol{\varepsilon}}_d$ is the rate of change of the deviatoric strain rate.

The evolution equation for the deviatoric stress ***s*** is

$$\frac{d\boldsymbol{s}}{dt} + \boldsymbol{s}\cdot\boldsymbol{\Omega} - \boldsymbol{\Omega}\cdot\boldsymbol{s} = -\frac{\boldsymbol{s}}{\tau_d} + 2G\dot{\boldsymbol{\varepsilon}}_d \quad ,$$

where $\boldsymbol{\Omega} = \frac{1}{2}\left[\nabla\tilde{\boldsymbol{u}} - (\nabla\tilde{\boldsymbol{u}})^T\right]$ is the spin tensor, $\tilde{\boldsymbol{u}}$ is the average velocity, τ_d is the relaxation time for deviatoric stress, $\dot{\boldsymbol{\varepsilon}}_d$ is the deviatoric strain rate, and G is the shear modulus.

Density is calculated using the same equation of state as in the `Linear` model.

The `Maxwell` parameters specified by user input in the data-entry window are as follows:

YoungModulus: real, default value = 0.0; Young's modulus.

PoissonRatio: real, default value = 0.0; Poisson's ratio.

bulkModulus: real, default value = 0.0; bulk modulus.

shearModulus: real, default value = 0.0; shear modulus of elasticity.

Note: If both **YoungModulus** and **PoissonRatio** are entered, the code internally calculates **bulkModulus** and **shearModulus** from their values. Otherwise, if both **bulkModulus** and **shearModulus** are entered, the code internally calculates **YoungModulus** and **PoissonRatio** from their values.

GruneisenCoefGammas: real, default value = 0.0; Gruneisen coefficient.

RelaxationTimeForPressure: real, default value = 1.0 x 10^{99}; relaxation time for pressure.

RelaxationTimeForDeviatoricStress: real, default value = 1.0 x 10^{99}; relaxation time for deviatoric stress.

Note: With the default values of **RelaxationTimeForPressure** and **RelaxationTimeForDeviatoricStress**, the `Maxwell` model is purely elastic.

.

failure: keyword text, default value = `none`; failure model, either `ductile`, `brittle`, or `none`.

Note: If a failure model is used, the **Damage** checkbox in the **Particle Properties** panel must be checked for the particle phase (Section 4.10).

elasticStressLim: real, default value = 9.5 x 10^9; elastic stress limit.

EOSa: real; A-term in equation of state (see `Linear` model).

EOSb: real; B-term in equation of state (see `Linear` model).

EOSc: real; C-term in equation of state (see `Linear` model), only used when energy transport is solved.

EOSd: real; D-term in equation of state (see `Linear` model), only used when energy transport is solved.

CpRef: real; specific heat at constant pressure.

CvRef: real; specific heat at constant volume.

pressureWork: real; There is a term for total derivative of pressure in the energy equation with enthalpy as the variable. Setting **pressureWork** to one enables this term. This should normally not be used.

HForm: real; enthalpy of formation. In general, the enthalpy of a material is expressed as: enthalpy = **HForm** + Cp(temperature – **TForm**), where Cp is the specific heat at constant pressure. See input parameters **TForm** and **CpRef**.

TForm: real; temperature of formation. See input parameter **HForm**.

Bsq: real; Boussinesq beta parameter. If **Bsq** > 1.0x10^{-32}, the Boussinesq approximation is used; this is for buoyancy-driven flow in cases where density differences are negligible, except where they appear in gravity terms.

Csp: real; close pack sound speed.

Thsp: real; volume fraction at close pack, also used in the calculation of the effective viscosity in the stress logic.

KRef: real; thermal conductivity.

Darcy: real; Darcy K-factor for interphase drag. Normally, **Darcy** is set to zero. If its value is set to a large value, say, 1.0x10^{30}, then it essentially fixes the phase (with zero velocity).

Note: A single Darcy factor is used for each phase; the value from the highest-numbered species in a phase will be used for that phase.

`Kelvin` (Kelvin-Voigt model)

Kelvin-Voigt materials can be considered as a viscous damper (dashpot) in parallel with an elastic spring. The CartaBlanca Kelvin-Voigt model is referred to simply as `Kelvin` in the input specifications.

In the Kelvin-Voigt model, the stress $\boldsymbol{\sigma}$ is separated into an elastic part $\boldsymbol{\sigma}_E$ and a viscous part $\boldsymbol{\sigma}_V$,

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_V + \boldsymbol{\sigma}_E \ .$$

The viscous part $\boldsymbol{\sigma}_V$ is calculated as

$$\boldsymbol{\sigma}_V = \mu_b \, tr(\dot{\boldsymbol{\varepsilon}})\boldsymbol{I} + 2\mu\dot{\boldsymbol{\varepsilon}} \quad ,$$

where μ_b is the bulk viscosity, μ is the shear viscosity and $\dot{\boldsymbol{\varepsilon}}$ is the strain rate.

The elastic part is calculated by solving the following evolution equation

$$\frac{d\boldsymbol{\sigma}_E}{dt} + \boldsymbol{\sigma}_E \cdot \boldsymbol{\Omega} - \boldsymbol{\Omega} \cdot \boldsymbol{\sigma}_E + \frac{1}{2} tr(\dot{\boldsymbol{\varepsilon}})\boldsymbol{\sigma}_E = B \, tr(\dot{\boldsymbol{\varepsilon}})\boldsymbol{I} + 2G\dot{\boldsymbol{\varepsilon}} \quad ,$$

where *B* is the bulk modulus, *G* is the shear modulus and $\dot{\boldsymbol{\varepsilon}}$ is the strain rate. The last term on the left hand side is necessary to ensure energy conservation by accounting for the effect of volume change in cases of large deformation (see the CartaBlanca Theory Manual [12] for additional details).

Density is initialized according to the equation of state

$$\rho^0 = A \, e^{((p/B_m)-(C(T-D)))} \quad .$$

where

ρ^0 = material density,

A, *B*$_m$, *C* and *D* are model parameters specified by user input in the data-entry window,

A	=	**EOSa**,
B_m	=	material bulk modulus, **bulkModulus**,
C	=	**EOSc**,
D	=	**EOSd**,

and

p = pressure, and
T = temperature.

When energy transport is not solved (see Section 4.2), the density is initialized as

$$\rho^0 = A e^{p/B_m} \ .$$

The `Kelvin` parameters specified by user input in the data-entry window are as follows:

YoungModulus: real, default value = 0.0; Young's modulus.

PoissonRatio: real, default value = 0.0; Poisson's ratio.

bulkModulus: real, default value = 0.0; bulk modulus.

shearModulus: real, default value = 0.0; shear modulus of elasticity.

Note: If both **YoungModulus** and **PoissonRatio** are entered, the code internally calculates **bulkModulus** and **shearModulus** from their values. Otherwise, if both **bulkModulus** and **shearModulus** are entered, the code internally calculates **YoungModulus** and **PoissonRatio** from their values.
.
bulkViscosity: real; bulk viscosity.

shearViscosity: real; shear viscosity.

failure: keyword text, default value = `none`; failure model, either `ductile`, `brittle`, `penhanced`, or `none`.

Note: If a failure model is used, the **Damage** checkbox in the **Particle Properties** panel must be checked for the particle phase (Section 4.10).

Note: Currently the `penhanced` option is not implemented for 3D.

elasticStressLim: real, default value = 9.5 x 10^9; elastic stress limit.

EOSa: real; A-term in equation of state.

EOSc: real; C-term in equation of state, only used when energy transport is solved.

EOSd: real; D-term in equation of state, only used when energy transport is solved.

CpRef: real; specific heat at constant pressure.

CvRef: real; specific heat at constant volume.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

KRef: real; see `Maxwell` parameters.

Darcy: real; see `Maxwell` parameters.

JohnsonCook

The Johnson-Cook model [4] adds a plasticity model to the Kelvin-Voigt model. Stress calculation for this constitutive model contains two parts, an elastic part and a plastic flow part.

The elastic stress increases as in the Kelvin-Voigt model. The plastic flow part starts by calculating the yield stress σ_{eq} as

$$\sigma_{eq} = \left(Y_0 + B_{jc}\varepsilon_p^n\right)\left[1 + C\ln\left(\dot{\varepsilon}_p \big/ \dot{\varepsilon}_0\right)\right]\left[1 - \left(\frac{T - T_0}{T_m - T_0}\right)^m\right] \quad ,$$

where C, n, m, Y_0 and B_{jc} are material parameters and $\dot{\varepsilon}_0$ is the characteristic strain rate, T is the temperature, T_m is the melting temperature, T_0 is the reference temperature and ε_p and $\dot{\varepsilon}_p$ are the effective plastic strain and the rate of the plastic strain. For many practical applications with large deformation, CartaBlanca approximates the effective plastic strain by an effective strain ε_e. The rate of the effective strain is calculated as

$$\begin{aligned}\dot{\varepsilon}_e &= \sqrt{2\left[(\dot{\varepsilon}_1 - \dot{\varepsilon}_2)^2 + (\dot{\varepsilon}_2 - \dot{\varepsilon}_3)^2 + (\dot{\varepsilon}_3 - \dot{\varepsilon}_1)^2\right]/9} \\ &= \sqrt{2\left|2[tr(\dot{\boldsymbol{\varepsilon}})]^2 - 6(\dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xx}\dot{\varepsilon}_{zz} + \dot{\varepsilon}_{yy}\dot{\varepsilon}_{zz} - \dot{\varepsilon}_{xy}\dot{\varepsilon}_{xy} - \dot{\varepsilon}_{xz}\dot{\varepsilon}_{xz} - \dot{\varepsilon}_{yz}\dot{\varepsilon}_{yz})\right|/9} \quad ,\end{aligned}$$

where $\dot{\varepsilon}_1$, $\dot{\varepsilon}_2$ and $\dot{\varepsilon}_3$ are the three principal rates of strain, and the $\dot{\varepsilon}$'s with double subscripts are the components of the rates of strain under the coordinate system used. The effective strain is the time integration of this effective strain rate. The effective stress σ_e is calculated similarly as

$$\begin{aligned}\sigma_e &= \sqrt{2\left[(\sigma_1 - \sigma_2)^2 + (\sigma_2 - \sigma_3)^2 + (\sigma_3 - \sigma_1)^2\right]/9} \\ &= \sqrt{2\left|2[tr(\boldsymbol{\sigma})]^2 - 6(\sigma_{xx}\sigma_{yy} + \sigma_{xx}\sigma_{zz} + \sigma_{yy}\sigma_{zz} - \sigma_{xy}\sigma_{xy} - \sigma_{xz}\sigma_{xz} - \sigma_{yz}\sigma_{yz})\right|/9} \quad ,\end{aligned}$$

where σ_1, σ_2, and σ_3 are the three principal stress rates, and the σ's with double subscripts are the components of the stress under the coordinate system used. If the effective stress σ_e is greater than the yield stress calculated from (4.12) then each deviatoric component of the stress is reduced by a factor σ_{eq}/σ_e to make the effective stress equal to the yield stress σ_{eq}.

The pressure, or the isotropic component of the stress, is kept unaltered in this step.

Density is initialized according to the equation of state

$$\rho^0 = \frac{A + Bp}{1 + C(T - D)} \quad ,$$

where

ρ^0 = material density,

A, B, C and D are model parameters specified by user input in the data-entry window,

A	=	**EOSa**,
B	=	**EOSb**,
C	=	**EOSc**,
D	=	**EOSd**,

and

p = pressure, and
T = temperature.

When energy transport is not solved (see Section 4.2), the equation of state is initialized as

$$\rho^0 = A e^{p/B_m} \quad ,$$

where B_m = material bulk modulus, **bulkModulus**.

The `JohnsonCook` parameters specified by user input in the data-entry window are as follows:

YoungModulus: real, default value = 0.0; Young's modulus.

PoissonRatio: real, default value = 0.0; Poisson's ratio.

bulkModulus: real, default value = 0.0; bulk modulus.

shearModulus: real, default value = 0.0; shear modulus of elasticity.

Note: If both **YoungModulus** and **PoissonRatio** are entered, the code internally calculates **bulkModulus** and **shearModulus** from their values. Otherwise, if both **bulkModulus** and **shearModulus** are entered, the code internally calculates **YoungModulus** and **PoissonRatio** from their values.
.
bulkViscosity: real; bulk viscosity.

shearViscosity: real; shear viscosity.

Note: The default values for the following eight parameters (**A**, **B**, **n**, **C**, **m**, **thetam**, **sigmaFail**, and **enthalpy0**) are for 4340 steel.

A: real, default value = 0.792; Y_0-term in plastic flow yield stress calculation.

B: real, default value = 0.510; B_{jc}-term in plastic flow yield stress calculation.

n: real, default value = 0.26; n-term in plastic flow yield stress calculation.

C: real, default value = 0.014; C-term in plastic flow yield stress calculation.

m: real, default value = 1.03; m-term in plastic flow yield stress calculation.

thetam: real, default value = 1793.0 K; melting point, T_m-term in plastic flow yield stress calculation.

sigmaFail: real, default value = 2.0 x 10^{10}; failure stress used in plastic flow ductile failure model.

enthalpy0: real, default value = 1.8 x 10^9; initial enthalpy used in plastic flow ductile failure model.

failure: keyword text, default value = `none`; failure model, either `ductile`, `brittle`, or `none`.

Note: If a failure model is used, the **Damage** checkbox in the **Particle Properties** panel must be checked for the particle phase (Section 4.10).

Note: To use the Johnson Cook plastic flow logic, enter `ductile` for **failure**.

elasticStressLim: real, default value = 9.5 x 10^9; elastic stress limit.

EOSa: real; A-term in equation of state.

EOSb: real; B-term in equation of state, only used when energy transport is solved.

EOSc: real; C-term in equation of state, only used when energy transport is solved.

EOSd: real; D-term in equation of state, only used when energy transport is solved.

CpRef: real; specific heat at constant pressure.

CvRef: real; specific heat at constant volume.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; temperature of formation, also reference temperature (T_0-term) in yield stress model.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

KRef: real; see `Maxwell` parameters.

Darcy: real; see `Maxwell` parameters.

`Tepla` (tensile plasticity model)

Note: The CartaBlanca implementation of the Tepla model is currently under development.

The CartaBlanca `Tepla` model is an implementation of the equations for plastic deformation with porosity growth given by Addessio et al. [1], which were based on the original TEPLA-F model of Johnson and Addessio [2, 3] for tensile plasticity and void growth in ductile fracture under general tensile loading conditions. In the following we use the notation of Addessio et al.

The code calculates a plastic yield stress $\boldsymbol{\tau}^p$ according to

$$\boldsymbol{\tau}^p = Y_s^p \sqrt{Y^\phi} \quad ,$$

where Y_s^p is the plastic flow stress ("no-void" yield stress) and Y^ϕ is the degradation of the strength of the material as a result of porosity growth. Currently Y_s^p is calculated with the Johnson-Cook model [4]:

$$Y_s^p = \left(c_1 + c_2 \left(\varepsilon_s^p\right)^n\right)\left[1 + c_3 \ln\left(\dot{\varepsilon}_s^p \big/ \dot{\varepsilon}_0\right)\right]\left[1 - \left(\frac{T - T_0}{T_m - T_0}\right)^m\right] \quad ,$$

where c_1, c_2, c_3, n, and m are user-input material parameters, $\dot{\varepsilon}_0$ is a user-input characteristic (reference) strain rate, T is the temperature, T_m is the user-input melting temperature of the material, T_0 is a user-input reference temperature, and ε_s^p and $\dot{\varepsilon}_s^p$ are the equivalent plastic strain and the equivalent plastic strain rate in the solid material. Currently, CartaBlanca sets the equivalent plastic strain rate equal to the reference strain rate. The strength degradation factor Y^ϕ is

$$Y^\phi = 1 + (q\phi)^2 - 2q\phi \cosh\left(-\frac{3P}{2Y_0}\right) \quad ,$$

where q and Y_0 are user-input material constants, ϕ is the porosity, and P is the pressure.

The code calculates an elastic effective stress using CartaBlanca's `Maxwell` model (with its relaxation times set to their defaults, giving a purely elastic result), and compares that stress with the plastic yield stress $\boldsymbol{\tau}^p$. If the effective elastic stress is less than or equal to $\boldsymbol{\tau}^p$, the stress and pressure are updated according to the elastic-Maxwell calculation. If the plastic yield stress is exceeded, the code uses equations (IV.4), (IV.5), and (IV.6) of Addessio et al. [1] to find a new porosity, including the relation in equation IV.5

$$\boldsymbol{\tau}^2 - \left(Y_s^p\right)^2 Y^\phi = 0 \quad ,$$

where $\boldsymbol{\tau}$ is the von Mieses stress and, again, Y_s^p and Y^ϕ are the plastic flow stress and the strength degradation factor, respectively. The code then uses equation (IV.4) of [1] to update the pressure, and equations (IV.5) and (IV.6) to update the stress tensor and to find a new equivalent plastic strain. In addition to the user-input parameters given above, the logic for the plastic regime uses the material's bulk modulus, shear modulus, and a Gruneisen coefficient for the material. The Gruneisen coefficient is taken directly from user-input. The bulk modulus and shear modulus are calculated internally from user-input values for the Young's modulus and Poisson ratio.

Note that the `Tepla` model's elastic-`Maxwell` calculation also uses a bulk modulus and a Gruneisen coefficient; the bulk modulus is also calculated from the `Tepla`-input Young's modulus and Poisson ratio, and the Gruneisen coefficient is used directly from the `Tepla` input.

CartaBlanca's `Tepla` model uses the input parameters EOSa and EOSb to initialize the density. Density is initialized according to

$$\rho^0 = A + Bp \quad ,$$

where

ρ^0 = material density,

A and B are model parameters specified by user input in the data-entry window,

A	=	**EOSa**,
B	=	**EOSb**,

and

p = pressure.

This expression is used whether or not energy transport is solved.

The `Tepla` input parameters are as follows:

Note: There is no failure model implemented in the `Tepla` model.

EOSa: real; constant term (A) in density initialization.

EOSb: real; pressure multiplier (B) in density initialization.

YoungModulus: real; Young's modulus.

PoissonRatio: real; Poisson ratio.

CpRef: real; see `Maxwell` parameters.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

KRef: real; see `Maxwell` parameters.

Darcy: real; see `Maxwell` parameters.

c1: real; material parameter in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook).

c2: real; material parameter in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook).

c3: real; material parameter in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook).

nyield: real; material parameter in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook). Equivalent plastic strain exponent (n).

myield: real; material parameter in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook). Exponent m of ratio of temperature differences.

strainRate0: real; characteristic (reference) strain rate, used in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook).

temp0: real; reference temperature, used in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook).

meltTemp: real; melting temperature, used in plastic flow stress ("no-void" yield stress) calculation (Johnson-Cook).

qDagradation: real; constant porosity multiplier in strength degradation calculation.

Y0: real; constant in strength degradation calculation (cosh term).

GruneisenCoefGammas: real; Gruneisen coefficient.

`Sesame` table

Note: The CartaBlanca implementation of the SESAME Library is currently under development.

The CartaBlanca `Sesame` constitutive model is an implementation of the Los Alamos SESAME Equation-of-State and Materials Properties Library [10].

`FortranModelOne`

`FortranModelOne` is a placeholder for any model written in Fortran.`

`GammaGas`

The `GammaGas` model is only used when energy transport is solved. Density is calculated according to

$$\rho^0 = \frac{A}{(\gamma - 1)\left(h/p - 1 \right)} \quad ,$$

where

ρ^0 = material density,

A is a model parameter specified by user input in the data-entry window,

A = **EOSa**,

γ is the ratio of specific heats (constant pressure to constant volume), set internally to1.4, for an ideal diatomic gas (a good approximation for air at standard conditions),

and

h = enthalpy,

and

p = pressure.

The `GammaGas` input parameters are as follows:

EOSa: real; A-term in equation of state.

EOSb: real; not used.

EOSc: real; not used.

EOSd: real; not used.

CpRef: real; see `Maxwell` parameters.

pressureWork: real; see `Maxwell` parameters.

HForm: real; see `Maxwell` parameters.

TForm: real; see `Maxwell` parameters.

Bsq: real; see `Maxwell` parameters.

Csp: real; see `Maxwell` parameters.

Thsp: real; see `Maxwell` parameters.

shearViscosity: real; see `Kelvin` parameters.

KRef: real; see `Maxwell` parameters.

Darcy: real; see `Maxwell` parameters.

ReferenceTemperature: real; see `NobleAbel` parameters.

`ViscousSolid`

The `ViscousSolid` model is currently only used for testing code numerics.

5. RUNNING CartaBlanca

A CartaBlanca calculation requires files `inputSpecifier.IO`, `NodeDataFile`, and `MeshFile`, and, optionally, `MeshPartitionFile` (required for parallel runs), `ParticleFile`, `BoundaryFile`, and `InitialConditionsFile`. The code is launched by running its Java class

```
gov.lanl.cartablanca.main.PhysMain
```

This can be done from the Unix (or Windows) command line, where a Unix script (or Windows `.cmd`) file is helpful, or from within an IDE (such as JBuilder, NetBeans, or Eclipse). There are two optional arguments for running `PhysMain`, which must be in order: (1) the name of the input-specifier file, and (2) the output-directory (1 must be included if 2 is to be used).

The calculation will run to a normal completion if either of the **General Information** parameters **Maximum Time** or **Maximum Cycles** is exceeded. An abnormal termination will happen if the code must reduce the time step size below **General Information** parameter **Minimum Time Step**; such a termination does not necessarily indicate a fundamental error (in code or model), but perhaps only that the current physical conditions require a smaller step size for the given mesh and transport algorithm(s). The status of a running calculation, including time step size, is sent in periodic edits to standard output (see Section 6.1). A discussion on CartaBlanca's time step control logic and suggestions for the time step user input are given in Section 7.2. Section 6.2 describes the code's major output (graphics) files.

Note that the GUI has two new toolbar buttons, "Run" and "STOP", which can be used to run/stop a calculation from within the GUI (see Section 4).

Two important code-running capabilities are Dump/Restart, which can add a great deal of flexibility to an analysis, and parallel computation, which can dramatically speed-up runtime. These are described in Sections 5.1 and 5.2, respectively.

5.1. Dump/Restart Capability

Often it may be desirable to break up a very long calculation into a series of two or more shorter runs. Also, one may wish to run a set of parametric studies from a common branch state, or restart a run with a smaller time step size. Or, one may simply wish to "see what happens" by extending a given calculation. CartaBlanca provides this capability by periodically writing binary dump files that capture a calculation's state at the problem time of a dump; these dump files can be used to restart a run.

.

The restart dump files are written to directory `output`; they are of the form

`dump.N.MMMMM.dfl` for ALE-mesh-specific state data, and

`dump.gridPhaseI.N.MMMMM.dfl` for PIC/MPM (particle)-specific state data, where `N` is an integer that specifies the partition (`0` for serial runs), `MMMMM` is a running sequence number that identifies the edit (with leading `0`'s), and `I` is an integer that identifies the particle-material (phase). For example, directory `output` would contain the following files from a run that had 80 edits, 4 partitions, and 2 particle materials:

```
dump.0.00000.dfl
dump.0.00001.dfl
…
…
…
dump.0.00080.dfl
…
…
…
dump.3.00000.dfl
…
…
…
dump.3.00080.dfl
dump.gridPhase2.0.00000.dfl
…
…
…
dump.gridPhase2.0.00080.dfl
…
…
…
dump.gridPhase2.3.00000.dfl
…
…
…
dump.gridPhase3.3.00080.dfl
```

where the particle-materials (phases) are materials 2 and 3 in this particular calculation.

CartaBlanca writes restart dump files according to a dump interval that is specified by the GUI **General Information** panel's Running Parameters **Graphics Time Interval** and **Graphics/Binary Dump Ratio**, where the latter is a multiplier applied to **Graphics Time Interval**.

A restart run is specified with the GUI **General Information** panel, by checking the **Restart** checkbox and setting Running Parameter **initGraphic** to the desired dump sequence number (any leading 0's are not necessary). All relevant `dump` and `dump.gridPhaseI` files must be in directory `output` (i.e., files for the grid, for all particle materials, and for all partitions).

5.2. Parallel Runs

Parallel computation is built into the code; CartaBlanca is designed around Java's built-in multi-thread capability, where processes can be run simultaneously (or share a single CPU) and can communicate with each other, but are controlled from the same program. Both shared and distributed memory architectures are supported. To do parallel computation with either shared or distributed memory, a partition file must be provided; this file is specified in the **General Information** panel, where also the **usePartitions** checkbox must be checked (see Section 4.1). Setting up a partition file is the only requirement for CartaBlanca shared memory computation.

For distributed memory machines, the MPJ package [8] is used. (Previously, we used the JavaParty extension to Java; although Los Alamos no longer supports distributed CartaBlanca computation via JavaParty, it should not be difficult to reimplement it.) MPJ is an extension to standard Java; it can be downloaded from

http://dsg.port.ac.uk/projects/mpj/soft/download.php

Step by step instructions for MPJ installation and usage are available at

http://dsg.port.ac.uk/projects/mpj/soft/readme.php

Following are the steps for user `CBuser` to run CartaBlanca under MPJ on a UNIX platform, using the C shell (`csh`).

Add to `CBuser`'s `.cshrc` startup file

```
setenv MPJ_HOME ~/mpj-v0_23
setenv PATH $MPJ_HOME/bin:/home/CBuser/apache-ant-1.6.5/bin:$PATH
```

To install MPJ, un-tar the downloaded zip file (MPJ is already compiled, to recompile type "`ant`" in directory `$MPJ_HOME`).

To run CartaBlanca, we need to edit and recompile three CartaBlanca source files:

`src/gov/lanl/cartablanca/main/PhysMain.java` ,
`src/gov/lanl/cartablanca/comm/Communication.java` , and
`src/gov/lanl/cartablanca/mesh/GlobalMesh.java`

Near the top of each file, comment-out the line

`import gov.lanl.cartablanca.comm.mpi.*;` and un-comment the line

`import mpi.*;`

To recompile CartaBlanca, enter

```
javac  -d classes -cp jars/swing-layout-
1.0.jar:$MPJ_HOME/lib/mpj.jar  src/gov/lanl/cartablanca/*/*.java
src/gov/lanl/cartablanca/*/*/*.java
src/gov/lanl/cartablanca/*/*/*/*.java
```

where the `javac` command is all on one line.

CartaBlanca is run with the following steps:

Set the running directory to the present directory,

```
setenv runDir `pwd`
```

Generate a file named `machines` in the running directory that contains a list of the names of the cluster nodes, in a column, on which CartaBlanca will run. For the Los Alamos `epiphany` machine, using the `vi` editor, we enter

```
vi machines (in $runDir)
E01
E02
…
…
```

Make an MPJ-Daemon on each of the cluster nodes listed in file `machines`:

```
mpjboot machines
```

If this is successful, the UNIX list command

`ls -rtl $MPJ_HOME/bin/` should show files such as `MPJ-DaemonE02.pid`, etc., in directory `$MPJ_HOME/bin/` .

Run CartaBlanca in the background:

```
nohup mpjrun.sh -np 2  -sport 15002  -wdir $runDir -mx2048m -cp
classes:$MPJ_HOME/lib/mpj.jar gov.lanl.cartablanca.main.PhysMain
inputSpecifier > & outA &
```

where again the command (`nohup` here) is all on a single line.

After the calculation has run to completion, delete the MPJ-Daemons on the cluster nodes listed in file `machines`:

```
mpjhalt machines
```

The files `DaemonE02.pid`, etc., should no longer be in directory `$MPJ_HOME/bin/` .

6. CartaBlanca RESULTS

CartaBlanca provides two types of output periodically as a problem runs:

- short edits to the standard output stream; by default to the console/screen, or as redirected to a file. These edits give status reports on a run, such as the current problem time and time step size. Also, at the start of a run, detailed reports on the problem initialization are written, which can be useful for training and diagnostic purposes.

- major edits of the problem state, written as text files to directory `output`. These are intended mainly for graphics postprocessing of the problem's results.

Sections 6.1 and 6.2 describe these two types of CartaBlanca output.

6.1. Console Status Prints

As a calculation proceeds, CartaBlanca sends status messages to standard output. Before the problem starts running, details on the problem's initialization are written, such as for `testBulletPlate.IO`:

```
…
…
…
In GlobalMesh, MeshFile is E:\cartablanca\meshes\2D\QUADS\41nx41n_10\myMeshFile.txt

In GlobalMesh, NodeDateFile is E:\cartablanca\meshes\2D\QUADS\41nx41n_10\myNodeDataFile.txt

In GlobalMesh, PartitionFile is E:\cartablanca\meshes\2D\QUADS\41nx41n_10\myPartitionFile.txt

Coordinate system is cartesian

 done reading elements
…
…
…

From MaterialResponse    <<=== user material input for Aluminum target.

 gridPhaseNum: 1 number of Species: 1

Material properties, from GenericSpecieResponse

 eosA eosB eosC eosD speciesName  Mattype eosType Cp k Visc closePackSSpeed close:ackVfrac

 2.7 3.934E-12 1.0 1.0 Aluminum flipParticles Kelvin 1.0 0.0 500.0 0.0 0.0
…
…
…

InitializeFields: doing momentum transport   <<=== bullet starts at -500 m/s.

  init region: 0, U1 = 0.0, V1 = 0.0, U2 = 0.0, V2 = 0.0, U3 = 0.0, V3 = 0.0, P = 100.0

  init region: 1, U1 = 0.0, V1 = 0.0, U2 = 0.0, V2 = 0.0, U3 = 0.0, V3 = 0.0, P = 100.0

  init region: 2, U1 = 0.0, V1 = -50000.0, U2 = 0.0, V2 = -50000.0, U3 = 0.0, V3 = -50000.0, P =
100.0
```

```
  init region: 3, U1 = 0.0, V1 = -50000.0, U2 = 0.0, V2 = -50000.0, U3 = 0.0, V3 = -50000.0, P =
100.0
…
…
…
```

The standard output edits are written according to a problem time step interval that is specified by the GUI **General Information** panel's Running Parameter **printlnStep**. Included are the time step number (`n`), problem time (`t`), time step size (`dt`), and solver and Newton Krylov iterations. Also, reports on restart dumps (Section 5.1) and graphics edits (Section 6.2) are written. As `testBulletPlate.IO` starts up, runs, and terminates, we see:

```
Dumping to file E:\cartablanca\output\dump.0.00000.dfl

Just wrote E:\cartablanca\output\dump.0.00000.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase2.0.00000.dfl

Just wrote E:\cartablanca\output\dump.gridPhase2.0.00000.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase3.0.00000.dfl

Just wrote E:\cartablanca\output\dump.gridPhase3.0.00000.dfl

  n = 00010  t =  2.00000E-008  dt =  2.00000E-009,  (0)  <<=== printlnStep = 10.

  n = 00020  t =  4.00000E-008  dt =  2.00000E-009,  (0)

Dumping to file E:\cartablanca\output\dump.0.00001.dfl

Just wrote E:\cartablanca\output\dump.0.00001.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase2.0.00001.dfl

Just wrote E:\cartablanca\output\dump.gridPhase2.0.00001.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase3.0.00001.dfl

Just wrote E:\cartablanca\output\dump.gridPhase3.0.00001.dfl

Done in Partition 0

Time for executing the problem: 7595 milliseconds.

Grind Time is 225 microseconds/cycle/node
```

(`testBulletPlate.IO` uses solver `NLExplicit` for its pressure solution, and no iteration data are printed.)

The content of the standard output can be configured by input switches, such as the **Verbose** checkbox in the **Preconditioner** input (Section 4.5).

There are numerous error messages that can be printed for abnormal conditions, either during problem initialization (e.g., missing mesh file, or inconsistent specifications) or execution (e.g., time step size hits lower limit and must be reduced).

For long production runs, it is recommended that the standard output be redirected to a file, in order to ensure that a complete record exists for the problem.

6.2. Graphics Output Files

The major-edit graphics text files are written to directory `output`, with naming conventions similar to the binary restart files (Section 5.1). They are of the form

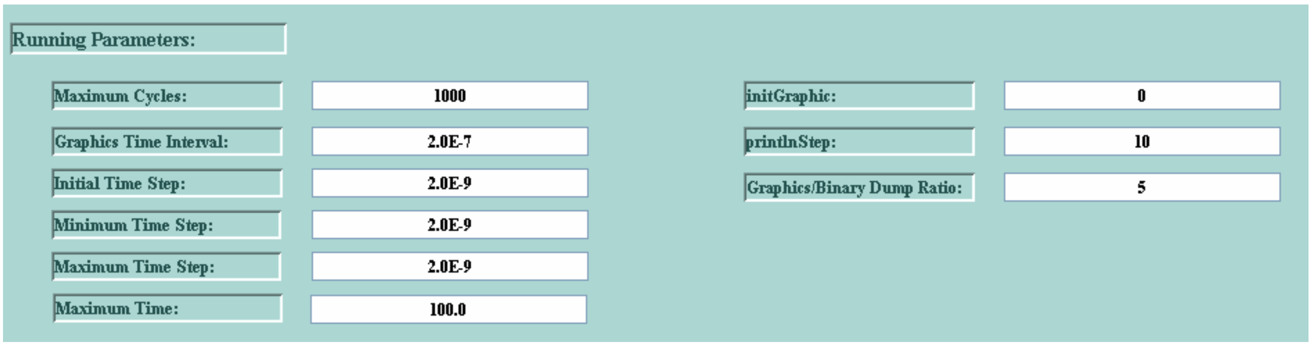
`plot.N.MMMMM.dat` for ALE-mesh-specific state data, and

`gridPhaseIpartitionN-MMMMM.dat` for PIC/MPM (particle)-specific state data,

where `N` is an integer that specifies the domain partition (0 for serial runs), `MMMMM` is a running sequence number that identifies the edit (with leading 0's), and `I` is an integer that identifies the particle-material (phase).

The graphics edits are written according to an edit interval that is specified by the GUI **General Information** panel's Running Parameter **Graphics Time Interval**.

Here we have set up the bullet-plate problem to run for 1000 time steps at a fixed step size = 2.0 x 10^{-9} s, with graphics edits (in Tecplot format) every 100 time steps (2.0 x 10^{-7} s):



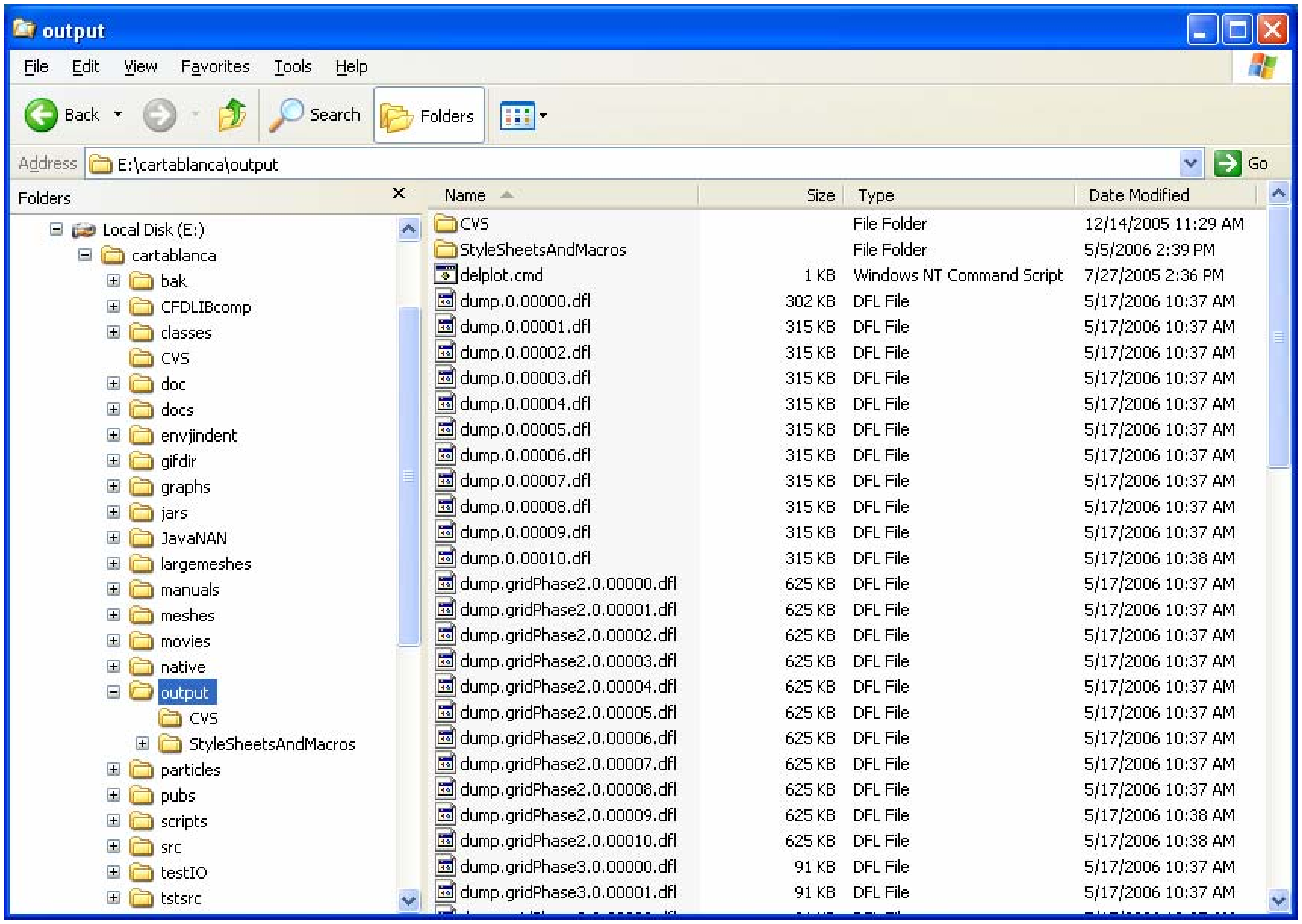
Running Parameters:

Maximum Cycles:	1000	initGraphic:	0
Graphics Time Interval:	2.0E-7	printlnStep:	10
Initial Time Step:	2.0E-9	Graphics/Binary Dump Ratio:	5
Minimum Time Step:	2.0E-9		
Maximum Time Step:	2.0E-9		
Maximum Time:	100.0		

The resulting graphics edits are written to directory `output`:

```
Graphics dump number 00009 at time =  1.80000E-006 cycleNumber = 900

  n = 00900  t =  1.80000E-006  dt =  2.00000E-009,  (0)

  n = 00910  t =  1.82000E-006  dt =  2.00000E-009,  (0)

  n = 00920  t =  1.84000E-006  dt =  2.00000E-009,  (0)

  n = 00930  t =  1.86000E-006  dt =  2.00000E-009,  (0)

  n = 00940  t =  1.88000E-006  dt =  2.00000E-009,  (0)

  n = 00950  t =  1.90000E-006  dt =  2.00000E-009,  (0)

  n = 00960  t =  1.92000E-006  dt =  2.00000E-009,  (0)

  n = 00970  t =  1.94000E-006  dt =  2.00000E-009,  (0)

  n = 00980  t =  1.96000E-006  dt =  2.00000E-009,  (0)

  n = 00990  t =  1.98000E-006  dt =  2.00000E-009,  (0)

Dumping to file E:\cartablanca\output\dump.0.00010.dfl

Just wrote E:\cartablanca\output\dump.0.00010.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase2.0.00010.dfl
```

```
Just wrote E:\cartablanca\output\dump.gridPhase2.0.00010.dfl

Dumping Particle Data to file E:\cartablanca\output\dump.gridPhase3.0.00010.dfl

Just wrote E:\cartablanca\output\dump.gridPhase3.0.00010.dfl

Graphics dump number 00010 at time =  2.00000E-006 cycleNumber = 1000

  n = 01000  t =  2.00000E-006  dt =  2.00000E-009,  (0)

Done in Partition 0

Time for executing the problem: 50302 milliseconds.

Grind Time is 29 microseconds/cycle/node
```

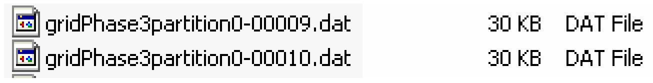


E:\cartablanca\output

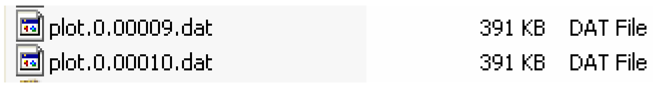
- Local Disk (E:)
  - cartablanca
    - bak
    - CFDLIBcomp
    - classes
    - CVS
    - doc
    - docs
    - envjindent
    - gifdir
    - graphs
    - jars
    - JavaNAN
    - largemeshes
    - manuals
    - meshes
    - movies
    - native
    - output
      - CVS
      - StyleSheetsAndMacros
    - particles
    - pubs
    - scripts
    - src
    - testIO
    - tstsrc

Name	Size	Type	Date Modified
CVS		File Folder	12/14/2005 11:29 AM
StyleSheetsAndMacros		File Folder	5/5/2006 2:39 PM
delplot.cmd	1 KB	Windows NT Command Script	7/27/2005 2:36 PM
dump.0.00000.dfl	302 KB	DFL File	5/17/2006 10:37 AM
dump.0.00001.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00002.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00003.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00004.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00005.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00006.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00007.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00008.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00009.dfl	315 KB	DFL File	5/17/2006 10:37 AM
dump.0.00010.dfl	315 KB	DFL File	5/17/2006 10:38 AM
dump.gridPhase2.0.00000.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00001.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00002.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00003.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00004.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00005.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00006.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00007.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00008.dfl	625 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase2.0.00009.dfl	625 KB	DFL File	5/17/2006 10:38 AM
dump.gridPhase2.0.00010.dfl	625 KB	DFL File	5/17/2006 10:38 AM
dump.gridPhase3.0.00000.dfl	91 KB	DFL File	5/17/2006 10:37 AM
dump.gridPhase3.0.00001.dfl	91 KB	DFL File	5/17/2006 10:37 AM

A total of 10 graphics dumps are written. For this case, the graphics files are of two types:



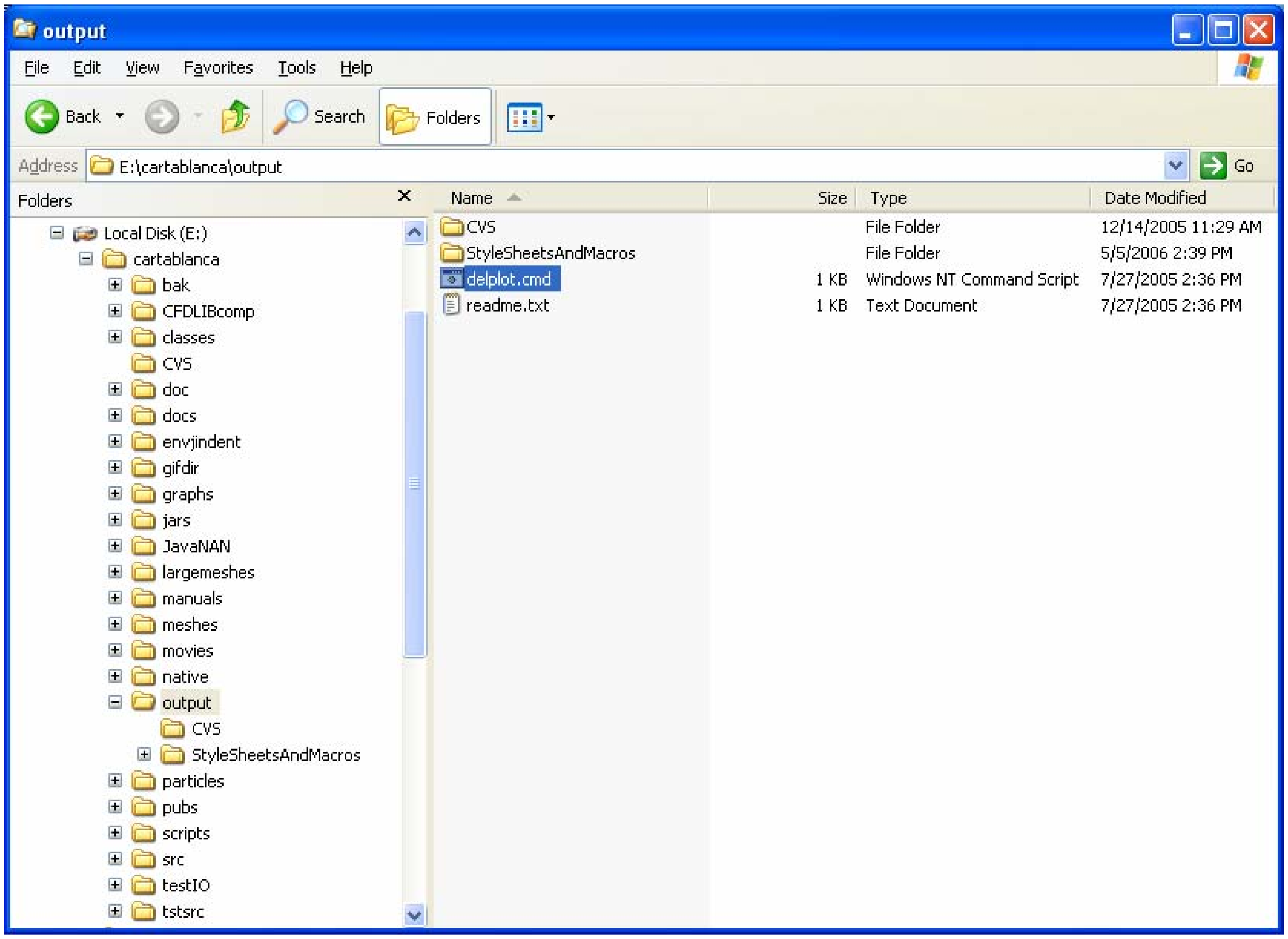
gridPhase3partition0-00009.dat	30 KB	DAT File
gridPhase3partition0-00010.dat	30 KB	DAT File



plot.0.00009.dat	391 KB	DAT File
plot.0.00010.dat	391 KB	DAT File

where the "`gridPhase`……" files contain graphics data that is "attached" to the individual particles that are used to model the bullet and the plate, and the "`plot`……" files contain data that is associated with the underlying Eulerian grid. A "`gridPhase`……" file is written and named for each particle-material, for each computational partition, and for each graphics edit. The "`plot`……" files are written for each partition and edit. (Note that dump/restart files are also written to directory `output`, with suffix `.dfl`)

The Windows command script `delplot.cmd` provides a convenient way to clean up directory `output`:



Name	Size	Type	Date Modified
CVS		File Folder	12/14/2005 11:29 AM
StyleSheetsAndMacros		File Folder	5/5/2006 2:39 PM
delplot.cmd	1 KB	Windows NT Command Script	7/27/2005 2:36 PM
readme.txt	1 KB	Text Document	7/27/2005 2:36 PM

6.2.1. ParaView-compatible Output

Graphics output that is compatible with the ParaView package is currently not available.

6.2.2. Time-History Plots

The standard CartaBlanca Tecplot-format output is a series of files, each of which contains state variable values at a specific time. CartaBlanca also contains a standalone postprocessing code that can read those files and produce a Tecplot time-history file for a user-specified CartaBlanca variable.

The time-history file is called `y.dat`, where "`y`" is the name of the dependent variable selected by the user; it is written to the directory that contains the original CartaBlanca Tecplot graphics files. The time-history code also writes a file called `y-summary.txt`, which contains diagnostic and other information for the current run.

A typical `.cmd` file that starts-up the time-history plot-file generator on Windows is

```
@echo off
rem
rem DOS batch file to run gov.lanl.cartablanca.graphics.XYPlotsFrame
rem

Title Console
java -mx512m -classpath E:\cartablanca\classes;E:\cartablanca\jars\swing-layout-1.0.jar
gov.lanl.cartablanca.graphics.XYPlotsFrame

pause
```

(where there is <u>no line-break</u> on the java command).

This brings up the frame shown in Figure 43.

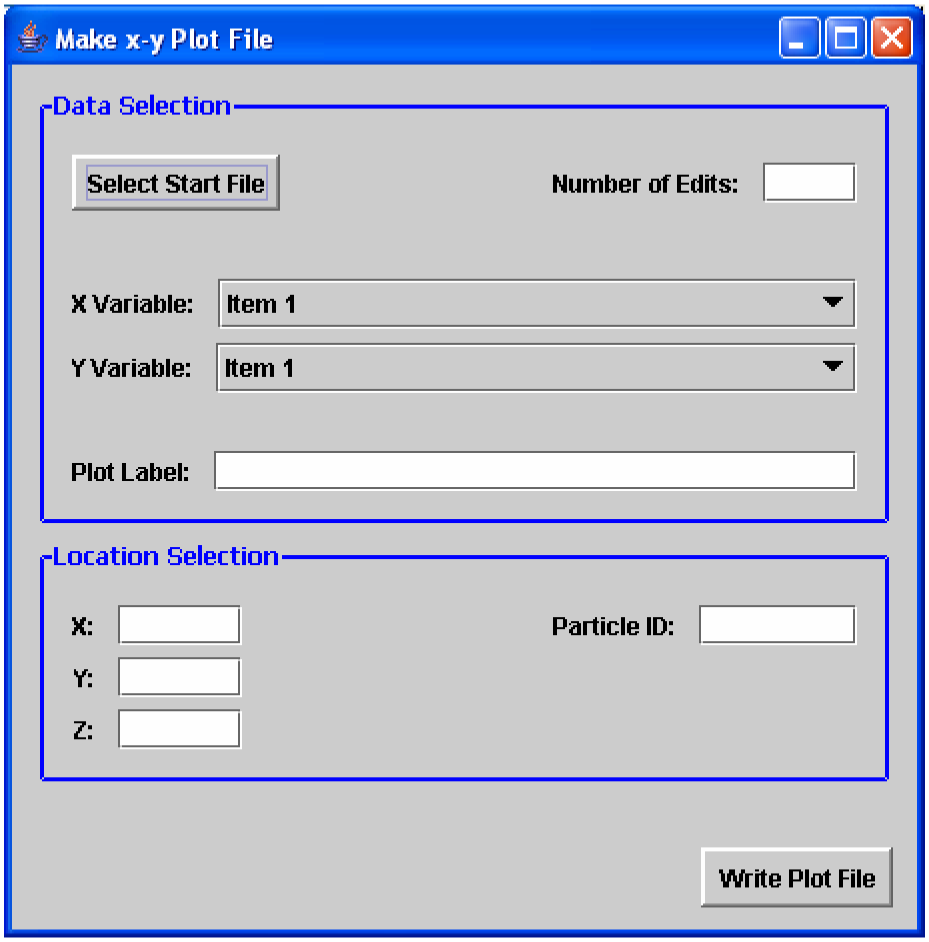


Figure 43. Frame for creating a time-history plot.

A "start file" must be specified; this is the first CartaBlanca time-edit for the time-history (x-y) plot you are making. Typically it would be a `gridPhaseNpartitionN-00000.dat` or `plot.N.00000.dat` file, or the initial file for a restart run, but you can start anywhere if you wish (splicing together multiple calculations from restarts is currently not supported). The button **Select Start File** shows a file chooser with file-type filter "`.dat`". To navigate your file system change "`.dat files`" to "`All Files`" in the chooser frame. At the desired directory you may wish to go back to "`.dat files`".

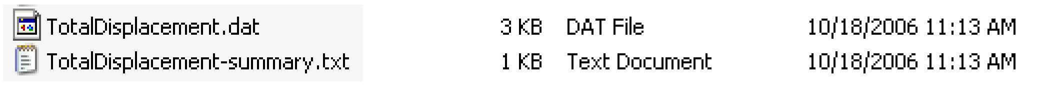
The number of consecutive CartaBlanca edits to include after the start file is entered, the dependent variable is selected from a dropdown list (currently only "time" is available for the x-axis), and an optional plot label is entered.

The location to plot is specified in the bottom portion of the frame. For particle files, an ID number (the PID in the original CartaBlanca Tecplot files) or (X, Y, Z) coordinates may be entered; if both are entered, PID will override the (X, Y, Z). Entering only (X, Y, Z) will result in the particle with initial coordinates closest to the entered (X, Y, Z) being followed. For Eulerian plot file data, the node closest to the entered (X, Y, Z) coordinates will be plotted.

The **Write Plot File** button generates the time-history `.dat` file and summary `.txt` file. A completion message is written to standard output:

```
Time-history file generator

***** Finished file E:\CB_output_files\3D\18Oct06\TotalDisplacement.dat
```

and the Tecplot-ready (x,y) plot will be in the directory that contains the original CartaBlanca graphics files:



```
TotalDisplacement.dat               3 KB   DAT File          10/18/2006 11:13 AM
TotalDisplacement-summary.txt       1 KB   Text Document     10/18/2006 11:13 AM
```

The summary file contains time and file stamps, coordinate information, and (for particle files) displacement information.

6.2.3. Animation

The Tecplot-format graphics output can be readily presented as movies using Tecplot macro files. Directory `output/StyleSheetsAndMacros` in the CartaBlanca distribution contains several sample macro files (with file extension "`.mcr`") that can be read by Tecplot to generate movies in `avi` format.

7. MODELING GUIDELINES

7.1. CartaBlanca Test Suite

The CartaBlanca test problems provide many examples of self-consistent models that can serve as training tools, and as the basis for further model development. They are organized in Java packages, according to the major physical process to be tested. Appendix A gives a detailed description of the test suite.

7.2. Time Step Size

In the input file, there are 3 time steps: Initial Time Step, Minimum Time Step and Maximum Time Step. For the NLExplicit solver, the flow system and energy system have a time step control to satisfy stability condition. For implicit solver, it can also reduce the time step to achieve convergence. Nevertheless, it is desirable to specify good choice of Initial Time Step. The Minimum Time Step is normally a small number like 1.0E-22 in our tests. However, if the time step is reduced to a very small number from the time step control and takes an unacceptable time to run, the user should increase the Minimum Time Step to stop the execution in the case. If NLExplicit is used, the user can do a rough estimate to find a reasonable Initial Time Step and Maximum Time Step. The user may also put in some number as the Initial and Maximum Time Step and run it for a few steps, let the code find the time step and then change the input.

7.3. Mesh and Particle Specification

7.4. Partitions for Parallel Computation

In order to achieve the best speedup for parallel calculations (i.e., to approach a speedup by a factor of N, where N is the number of processors), it is essential to balance the computational load across the parallel processors. For solid-structure calculations that use the Material Point Method (PIC method), the total number of computational particles may dominate the runtime; in such cases the user should be careful to distribute the particles in the computational domain evenly across the partitions. If the particle and mesh calculations both have significant impact on the runtime, it may be possible to exploit any symmetry in the problem to set up the partitions.

7.5. Guidelines for GUI Input Panels

Sections 7.5.1 – 7.5.11 give additional observations and recommendations on modeling options and input values, for each of the GUI input panels.

7.5.1. General Information

The binary restart dumps, although not especially time-consuming, can use up a lot of storage space, and typically not as many as the graphics edits will be needed. Therefore, it is usually a good idea to set **Graphics/Binary Dump Ratio** to, say, 5 or 10.

A good value for standard output (console output) Running Parameter **initGraphic** is 10 time steps. Also, it is a good idea to redirect the standard output edits to a file, to keep a permanent (and complete) record of a run; the console edits can be pasted into an editor, but they can easily overflow a window's text buffer.

7.5.2. Physics

Currently the Material Point Method in CartaBlanca does not support implicit time advancement. Therefore, when computational particles are used to track material interfaces (for example, for fluid-structure or structure-structure interactions) or history-dependent material effects, the **flowSystem** should be `NLMultiPhaseFlowPexp` (explicit in pressure).

7.5.3. Solver

7.5.4. Numerical Options

7.5.5. Preconditioner

The SSOR and ILU0 solver methods cannot be used for parallel calculations. The code will write an error message and shut down in this circumstance.

7.5.6. Initial Conditions

7.5.7. Boundary Conditions

7.5.8. Exchange Parameters

7.5.9. Chemical Reaction

7.5.10. Particle Properties

7.5.11. Species Properties

8. REFERENCES

[1] Addessio, F. L., Zuo, Q.H., Mason, T.A., and Brinson, L.C., A model for high-strain-rate deformation of Uranium-Niobium Alloys, Los Alamos National Laboratory report LA-14034-MS (2003).

[2] Gurson, A.L., Continuum theory of ductile rupture by void nucleation and growth: Part I-Yield criteria and flow rules for porous ductile media, ASME J. Eng. Mater. Technol. **99**, 2 (1977).

[3] Johnson, J.N. and Addessio, F.L., Tensile plasticity and ductile fracture, J. Appl. Phys. **64** (12), 6699 (1988).

[4] Johnson, G.R. and Cook, W.H., Fracture characteristics of three metals subjected to various strains, strain rates, temperatures and pressures, Eng. Fract. Mech. **21** (1), 31 (1985).

[5] JUnit link, http://www.JUnit.org/.

[6] Karypis, G. and Kumar, V., *METIS A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices, Version 4.0*, University of Minnesota, Department of Computer Science/Army HPC Research Center, Minneapolis, MN http://glaros.dtc.umn.edu/gkhome/views/metis/index.html (http://www-users.cs.umn.edu/~karypis/metis/index.html).

[7] Knoll, D.A., Prinja, A.K., and Campbell, R.B., A direct Newton solver for the two-dimensional Tokamak edge plasma fluid equations, J. Comp. Physics, **104**, 418 (1993).

[8] MPJ link, http://www.hpjava.org/papers/MPJ-CPE/cpempi/cpempi.html

[9] ParaView link, http://www.paraview.org/HTML/Index.html

[10] SESAME link, http://t1web.lanl.gov/newweb_dir/t1sesame.html

[11] Tecplot link, http://www.tecplot.com/index.htm

[12] Zhang, D.Z., VanderHeyden, W.B., Zou, Q., Ma, X., and Giguere, P. T., CartaBlanca Theory Manual: Multiphase Flow Equations and Numerical Methods, Los Alamos National Laboratory report LA-UR-07-3621 (2007).

[13] Zhang, D.Z., VanderHeyden, W.B., Zou, Q., Ma, X., and Giguere, P. T., CartaBlanca Programmer's Manual, Los Alamos National Laboratory report, to be published.

[14] Zou, Q., Zhang, D.Z., Padial-Collins, N.T., and VanderHeyden, W.B., Multiphase Flow Simulation of Ignition of Solid Explosive, Proceedings of the ICMF-2004, 5th International Conference on Multiphase Flow, ICMF'04, Yokohama, Japan, May 30-June 4, 2004, Paper No. 239.

APPENDIX A: CartaBlanca Test Suite

The CartaBlanca Test Suite comprises two sets of problems:

- 47 fast-running problems that are typically run with the JUnit testing framework [5] (see also Section 3.4). Each of these problems executes in a few seconds on desktop hardware.

- 5 longer-running problems that are typically run with batch scripts.

The Test Suite provides many examples of CartaBlanca use, and its problems can serve as the basis for development of new problems.

Note: parameter **physicsType**: The CartaBlanca `inputSpecifier.IO` input file includes a String parameter called **`physicsType`**. Parameter **`physicsType`** is used internally to set the high-level physics solution driver; its default value is `NLMultiPhysicsDriver`, and it is not part of the GUI. `NLMultiPhysicsDriver` is the most commonly used **physicsType**; it provides a general way to set up for solving the (multiphase) Navier-Stokes equations, and it controls CartaBlanca physics systems such as flow system `NLMultiPhaseFlowPexp` and its Java extensions, energy system `NLEnergyBasic` and its Java extensions, species system `NLSpeciesBasic` and its Java extensions, and turbulence system `NLTurbulence` (see Section 4.2 on the **Physics** panel). Normally, if the multiphase flow system is to be solved, **physicsType** should be `NLMultiPhysicsDriver`. Other **`physicsType`**s can be specified; these are used for special applications or were developed for early code testing. The Test Suite includes a few such cases, which are indicated below (of course the `inputSpecifier.IO` input file can always be edited by hand to use a non-default **physicsType**).

CartaBlanca is set up to automatically create `inputSpecifier.IO` input files for each of the fast-running problems, which are passed to the JUnit testing framework (see Section 3.4). In the following, each such file is identified by a unique `filename.IO` that is specified by the code. Four of the five longer-running problems can also be written by CartaBlanca methods; the fifth is maintained as a `.IO` file (see discussion below).

Fast-Running Problems

When JUnit is launched, each input file for the fast-running problems is automatically written by a separate Java routine (method); these methods are built-in to CartaBlanca, and are organized into Java packages according to the main physical process or code logic that is being tested (all are under `gov.lanl.cartablanca.test`). In the following we group the fast-running problems into their respective Java packages. All input files have extension `.IO`, and are written to directory `testIO`. (The Java classes (`.java` files) that write the input files typically have similar names, with `Test` appended.)

- `advection`: `AdvZigZagP`, `cadvexpl`, `nlcadvntngmres`, `nlscadvntngmres`, `onedcadvexp`, `scadvgmres`. Each is a test of advection, each with its own **physicsType**: `NLMultiPhysicsDriver`, `CompatibleAdvection`, `NLCompatibleAdvection`, `NLScalarAdvection`, `ScalarAdvectionOneD`, and `ScalarAdvection`, respectively. `AdvZigZagP` is the newest of the package; it tests advection to equilibrate pressure in the case where the initial pressures and densities are in a zig-zag pattern.

- `analyticsoln`: `Couette`, `Poiseuille1`, `PoiseuilleCylind`, `Poiseuille1_RF`. Test analytic solutions including Couette flow and Poiseuille flow in a 2-D region and in a 2-D cylindrical region. These tests are for single-phase viscous incompressible flows. They use `NLMultiPhysicsDriver` as the **physicsType**. These tests only solve the momentum equation. The **PeriodicInY** boundary condition is used (see Section 4.1.3). `Poiseuille1_RF` reads a boundary condition data file to set a boundary condition region.

- `energy`: `testNLEnergyBasic`, `testNLEnergyBasicWithFlowBasic`. Both problems use `NLMultiPhysicsDriver` as the **physicsType**; they solve the energy equation, without or with the momentum equation, with two phases, water and ice, treated as fluids. Both problems have the same geometry in a 2-D pipe, with cooling sections on the top and bottom boundaries, and have phase transition from water to ice. `testNLEnergyBasicWithFlowBasic` has a background flow in the pipe.

- `heattransfer`: `htpcg`, `htpgmres`, `htpgmres4thds`, `nlhtpntngmres`. Heat transfer cases. Each problem has its own **physicsType**.

- `miscellaneous`: `DisOps`, `DisOpsWithPeriodicity`, `DisOpsWPInTheta`, `Poisson_equation`. The first three problems test discrete operators in 3-D, using **physicsType** `DisOps`. `DisOps` and `DisOpsWithPeriodicity` use partitions. `DisOpsWithPeriodicity` uses **PeriodicInX**. `DisOpsWPInTheta` uses **PeriodicInTheta**. `Poisson_equation` tests for Maxwell-equations solution in the electrostatic limit by solving Poisson's equation in 2-D. The **physicsType** of `Poisson_equation` is `ESMaxwell`, and the **Solver1 Field** parameter is `Special`. Note that the Java code that writes `Poisson_equation.IO` is in file `MaxwellEquationsTest.java`.

- `mpflow`: these tests are multiphase flow cases, all use `NLMultiPhysicsDriver` as the **physicsType**, to solve the momentum equation only. They have either one or two (or in one case, three) incompressible fluid phases. They can be put in several groups:

  `BrokenDamTris1pBasic`, `BrokenDamTris2pBasic`, `DamComparison`. Broken dam tests with different geometries.

  `nlmultiphaseflowbasic`. A 2-D square box is filled with two incompressible inviscid fluids, the heavy fluid is on the left half and the light one is on the right, the motion is from gravity.

  `NLMultiPhaseFlowViscous1Basic`, `NLMultiPhaseFlowViscous1Impl`. Both are single phase incompressible viscous flow in a rectangular pipe [0, 5] by [0, 1], the fluid is at rest initially. The left side has an inflow boundary condition, with volume fraction and velocity specified, the right side has a pressure boundary condition with pressure specified. The fluid moves accordingly. They use flow systems `NLMultiPhaseFlowBasic` and `NLMultiPhaseFlowImpl`, respectively.

`NLMultiPhaseFlowViscous2Basic`, `NLMultiPhaseFlowViscous2ImplStress`, `NLMultiPhaseFlowViscous2Impl`. `NLMultiPhaseFlowViscous2Basic` has two-phase incompressible viscous flow in a rectangular pipe [0, 5] by [0, 1]. (However, the second fluid is set to an initial volume fraction of 0.0). The fluid is at rest initially. The left side has an inflow boundary condition, with volume fraction and velocity specified (the second fluid has volume fraction 0.0), the right side has a pressure boundary condition. The fluid moves accordingly. Along part of the top and bottom of the pipe, there are boundary conditions of type `wall`, with velocity set to zero ("sticky" boundaries). `NLMultiPhaseFlowViscous2ImplStress` and `NLMultiPhaseFlowViscous2Impl` have one fluid phase in a [0, 1] by [0, 1] grid; along the top ($Y = 1.0$) the X-velocity is set at 1.0 cm s^{-1} with a `wall` boundary condition. These problems use flow systems `NLMultiPhaseFlowBasic`, `NLMultiPhaseFlowImplicitStress`, and `NLMultiPhaseFlowImpl`, respectively.

`SmallHGLayerT4pBasic`, `SmallHourGlassT4pBasic`, `trickleBasic`. The first two problems have two-phase incompressible flow in an hourglass geometry; a heavy fluid is above a light fluid, and flow is from gravity. Triangle elements are used for the mesh. `trickleBasic` has three fluids with different densities in a rectangular grid with quadrilateral elements.

- <u>`particle`</u>: All use the MPM/PIC particle method (with the exception of problem `testParticlePiston`, as described below) with **physicsType** `NLMultiPhysicsDriver`, and use flow system `NLMultiPhaseFlowPexp` (explicit for solving the momentum equation), with two or more phases. Phases represented by particles are solid materials. A fluid phase (air, or other gas) is also present. In most cases, only the momentum equation is solved. In `testParticleWithFlowAndEnergyPexp` the momentum and energy equations are solved. In `testParticleWithFlowAndReaction1` and `testParticleWithFlowAndReaction2` momentum, energy, and species equations are solved, with chemical reaction. The following tests are in this group:

  `testBulletPlate`. A lead bullet penetrates an aluminum plate in a 2-D box with air in the background.

  `testParticleCylindrical`. A tungsten projectile penetrating a target in 2-D cylindrical geometry.

  `testParticleDamage`. A lead projectile penetrating a concrete target in a 2-D box, to test solid damage.

  `ParticleOps`. A simple case to test some basic particle operations.

  `testParticleTranslation`. A uniform translation of a solid phase in air in 1-D; uses `inflow` and `pressure` boundary conditions.

testParticleWithFlowAndEnergyPexp. Two phase case with one particle phase and one fluid phase. Both phases are present initially, with zero velocity and uniform temperature; the problem should remain steady.

testParticleWithFlowAndReaction1. One particle phase of high explosive and one fluid phase of a product gas. There is a simplified one-step chemical reaction from the explosive to the fluid phase.

testParticleWithFlowAndReaction2. Dump/restart test. Restarts the calculation from a dump file written by testParticleWithFlowAndReaction1 and continues for additional steps.

testParticleWithFlowPexp. A solid bar with initial velocity drops in air in a 2-D box.

testParticleWithFlowPexp4p. A parallel run of testParticleWithFlowPexp with 4 processors (it runs even if there is only one CPU).

testParticlePiston. Solves the momentum equation for a piston in gas, in 1-D, using the **solveStress** option (see **Physics** panel, Section 4.2). This is an alternative way to solve for stresses in solid materials, where only the ALE grid is used (particles are not used). Note that the current selection of **Use Particles** for the problem's ViscousSolid material is not necessary.

shortTungstenParallel. Two solid phases in air, using a partition file for parallel calculation. Note that the Java code that writes shortTungstenParallel.IO is in file ParticleParallelTest.java.

testParticleSinglePhaseTranslation. Translation of a solid phase in 1-D (no fluid is present); uses a single pressure boundary condition.

- species: nlspecies, nlmasstransfer. nlspecies uses the **solveSpeciesTransport** option (see **Physics** panel, Section 4.2), with a single, one-species, incompressible fluid. nlmasstransfer has two phases, with two species in each phase.

Longer-Running Problems

The five longer-running problems, which provide very detailed code tests, are typically not run as part of the JUnit testing, but rather by Los Alamos-developed batch scripts. Input files for four of the five longer-running problems are written by CartaBlanca methods, similar to the fast-running problems; the fifth input file, LongTungsten, is maintained separately as a .IO file that is read (and slightly modified) by a CartaBlanca Java method that creates the version that is tested. The five methods that create the five longer-running .IO input files are in package gov.lanl.cartablanca.test.particle.

The five longer-running problems are:

testLongSpalling. A spalling problem, in which a short plate impacts a long plate, and the combined stress wave breaks the long plate in the middle. A 2-D quadrilateral mesh is used.

`LongTungsten`. A tungsten projectile penetrating a target. A 2-D quadrilateral mesh is used. The basic input specifications are maintained in file `LongTungsten` .IO, in directory `testIO`.

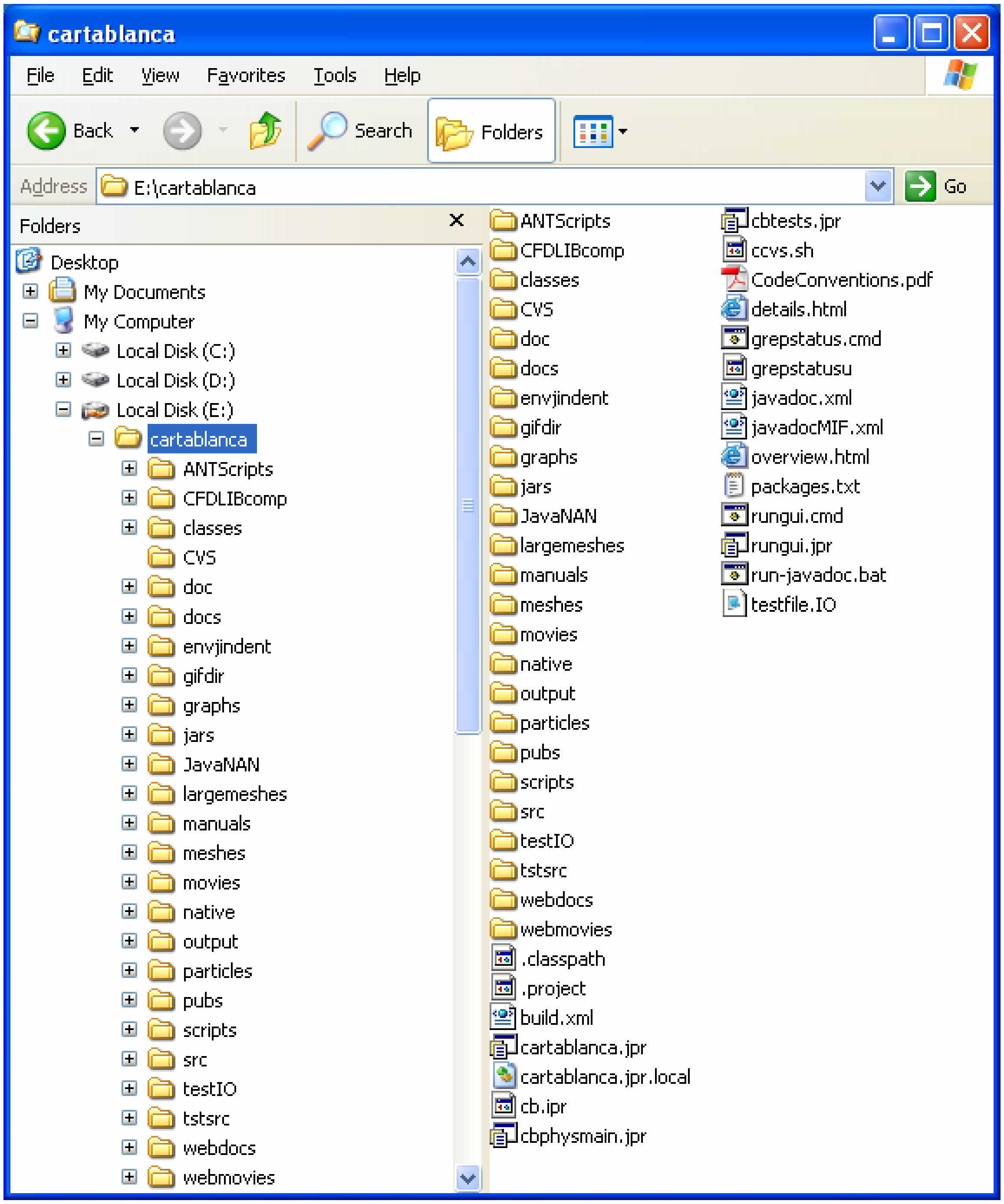
`testLongVibrationShell`. Tests a vibrating shell, formed by two concentric spheres. A polar coordinate shape QUADS mesh is used with a `cylindrical` coordinate system. Initially an inward radial velocity of 5000 cm s^{-1} is applied to the particles that comprise the shell. The resulting vibration period is compared to the theoretically expected value.

`testLongVibration`. Originally developed to test energy conservation over time. A 2-D quadrilateral mesh is used.

`testLongVibration1d`. Originally developed to test the convergence of the MPM/PIC method, in 1-D.

APPENDIX B: CartaBlanca Release Package

The top-level files and directories in the unzipped CartaBlanca release file are shown in Fig. 44.



```
Desktop
  My Documents
  My Computer
    Local Disk (C:)
    Local Disk (D:)
    Local Disk (E:)
      cartablanca
        ANTScripts
        CFDLIBcomp
        classes
        CVS
        doc
        docs
        envjindent
        gifdir
        graphs
        jars
        JavaNAN
        largemeshes
        manuals
        meshes
        movies
        native
        output
        particles
        pubs
        scripts
        src
        testIO
        tstsrc
        webdocs
        webmovies
```

```
ANTScripts            cbtests.jpr
CFDLIBcomp            ccvs.sh
classes               CodeConventions.pdf
CVS                   details.html
doc                   grepstatus.cmd
docs                  grepstatusu
envjindent            javadoc.xml
gifdir                javadocMIF.xml
graphs                overview.html
jars                  packages.txt
JavaNAN               rungui.cmd
largemeshes           rungui.jpr
manuals               run-javadoc.bat
meshes                testfile.IO
movies
native
output
particles
pubs
scripts
src
testIO
tstsrc
webdocs
webmovies
.classpath
.project
build.xml
cartablanca.jpr
cartablanca.jpr.local
cb.ipr
cbphysmain.jpr
```

Figure 44. CartaBlanca release directories and files (top level).

Directories that have been used in this document are:

`cartablanca`: The top-level directory. Includes JBuilder project files `rungui.jpr`, `cbphysmain.jpr`, and `cbtests.jpr`, that run the GUI, the main code, and the JUnit short-running test suite, respectively. By default, the GUI and main code will read file `inputSpecifier.IO` from this directory. Also has Windows/DOS command file `rungui.cmd` to run the GUI without the JBuilder interface.

`meshes`: A large collection of node, mesh, and partition files for 1-D, 2-D, and 3-D problems, organized in subdirectories according to the various geometries included. All meshes used by the test suites are included here. Typically, a user will put a new set of mesh files in a new subdirectory in the `meshes` hierarchy.

`output`: By default, the code will write dump and graphics files to directory `output`. Includes Windows/DOS command file `delplot.cmd`, which can be used (with care) to clean up the directory.

`particles`: Includes a single sample file that specifies particle coordinates for MPM/PIC input. Typically, the user will choose automatic particle generation in the **General Information** panel, specifying the number of particles in the **Particle Properties** panel.

`scripts`: Unix scripts, some of which run the GUI, the main code, and the JUnit short-running test suite.

`src`: Java source code for the main code and the GUI. Also has package `gov.lanl.cartablanca.main.generatemesh` and file `gov/lanl/cartablanca/graphics/XYPlotsFrame.java`.

`testIO`: The `.IO` files for the fast-running test problems are written here by the JUnit framework. Also has the main specification of the `LongTungsten` test problem.

`tstsrc`: Java source code to generate the fast-running and four of the longer-running test problems. Also has code that controls generation of the `LongTungsten` test problem. Uses code from directory `src`.