INL/EXT-13-29799

Multiphysics Integrated Coupling Environment (MICE) User Manual

August 2013

INL/EXT-13-29799

Multiphysics Integrated Coupling Environment (MICE) User Manual

August 2013

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Assistant Secretary for
Environmental Management
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

CONTENTS

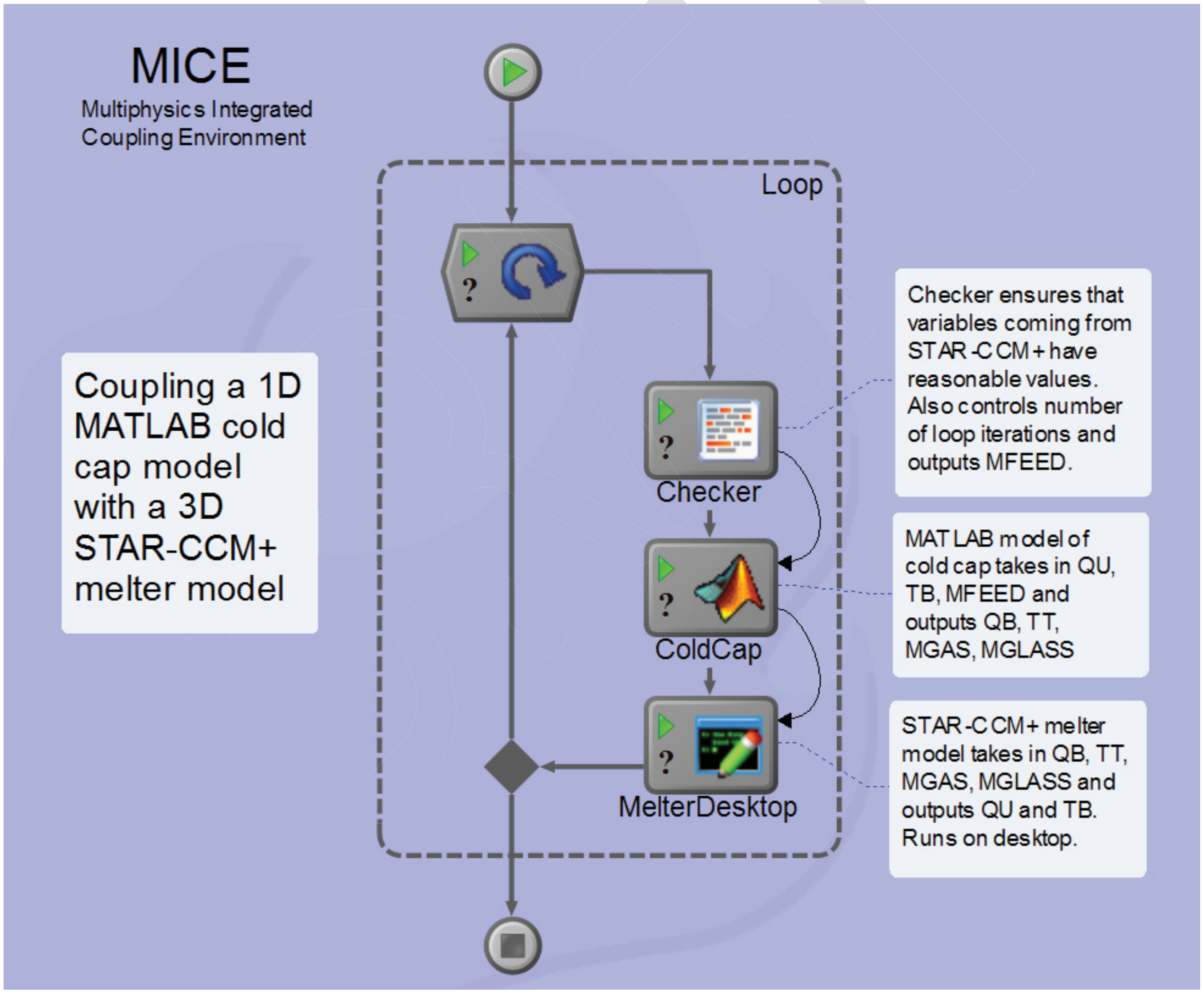
ACRONYMS

GUI	Graphical User Interface
HPC	High Performance Computing environment
MICE	Multiphysics Integrated Coupling Environment
ODE	ordinary differential equation

Multiphysics Integrated Coupling Environment (MICE)

1. INTRODUCTION

The complex, multi-part nature of waste glass melters used in nuclear waste vitrification poses significant modeling challenges. The focus of this project has been to couple a 1D MATLAB model of the cold cap region within a melter with a 3D STAR-CCM+ model of the melter itself. The Multiphysics Integrated Coupling Environment (MICE), shown below, has been developed to create a cohesive simulation of a waste glass melter that accurately represents the cold cap. The one-dimensional mathematical model of the cold cap uses material properties, axial heat, and mass fluxes to obtain a temperature profile for the cold cap, the region where feed-to-glass conversion occurs. The results from Matlab are used to update simulation data in the three-dimensional STAR-CCM+ model so that the cold cap is appropriately incorporated into the 3D simulation. The two processes are linked through ModelCenter integration software using time steps that are specified for each process. Data is to be exchanged circularly between the two models, as the inputs and outputs of each model depend on the other.


MICE
Multiphysics Integrated Coupling Environment
Coupling a 1D MATLAB cold cap model with a 3D STAR-CCM+ melter model
Loop
Checker
ColdCap
MelterDesktop
Checker ensures that variables coming from STAR-CCM+ have reasonable values. Also controls number of loop iterations and outputs MFEED.
MATLAB model of cold cap takes in QU, TB, MFEED and outputs QB, TT, MGAS, MGLASS
STAR-CCM+ melter model takes in QB, TT, MGAS, MGLASS and outputs QU and TB. Runs on desktop.

The coupling between the MATLAB and STAR-CCM+ models has been approached in two ways. The first approach is a programming-based approach, in which the two models exchange data over network sockets. The socket communications are encoded in Java, and the models are linked using Simulink, a graphical programming language. The second approach to the coupling is software-based: ModelCenter is used to create wrapped components for the MATLAB and STAR-CCM+ models and then link them together to allow for data exchange. Overall, it is clear that ModelCenter allows for significantly more functionality and usability than the programming approach implemented in Simulink.

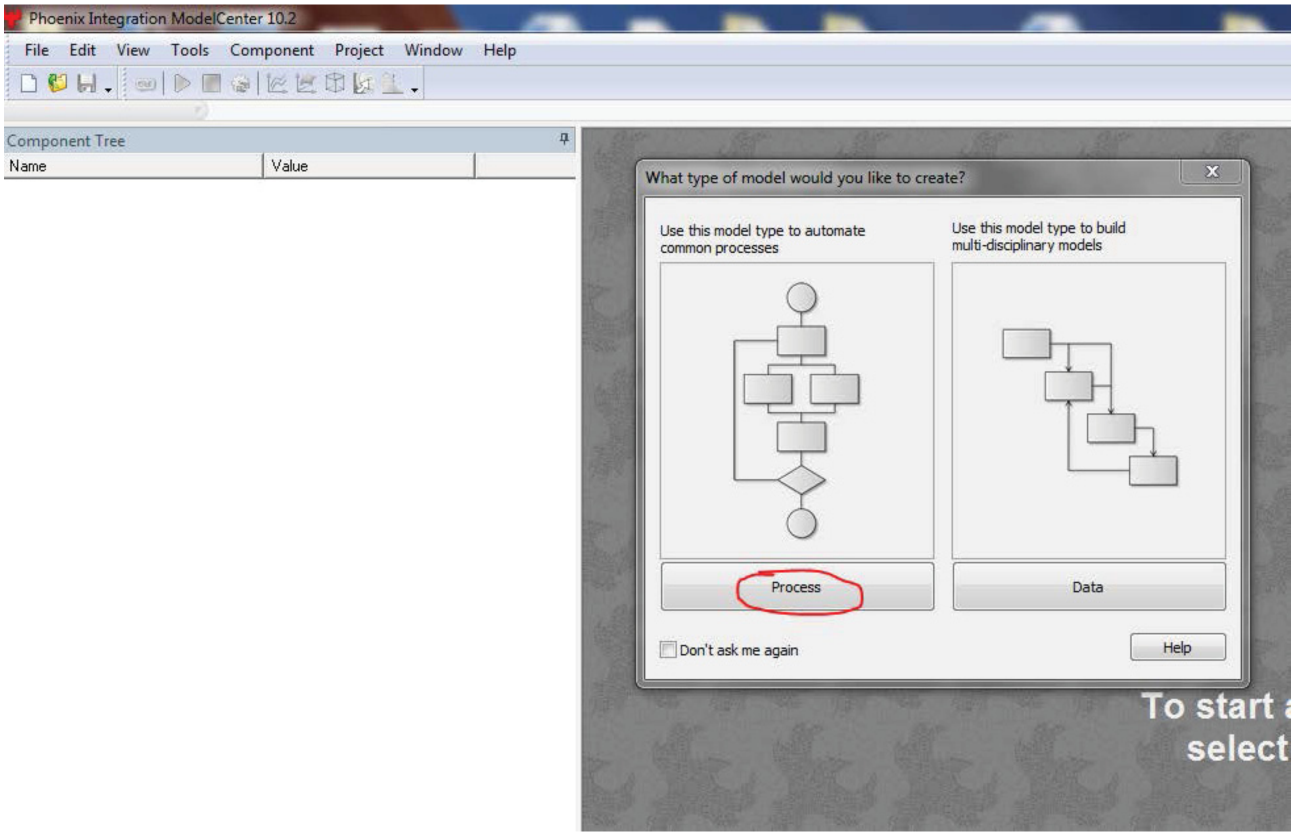
In the Simulink coupling, STAR-CCM+ is represented as a Level 2 MATLAB S-function, and the MATLAB code for the cold cap has been included directly, as Simulink offers direct integration of MATLAB code. Unfortunately, however, Simulink does not support some of the functionalities offered by MATLAB. Therefore, the original MATLAB model for the cold cap has been amended heavily to function properly in this coupling. To connect the MATLAB and STAR-CCM+ models, programming is used to establish network sockets and a framework for how information is transmitted between the two models. Currently, after weeks of effort, this coupling approach is still a work in progress – it has been much more difficult than originally anticipated to establish a connection between the two models using Java, even with an example of a similar coupling at our disposal. This brings into question the sustainability of this model: model maintenance would require significant expertise in Simulink, MATLAB and Java and also a working knowledge of socket communications. Almost all manipulations to this simulation involve programming, and very little documentation has been found for such a coupling.

The ModelCenter coupling, on the other hand, has been quite successful. ModelCenter offers an easy-to-use plug-in for MATLAB which can execute code externally in a MATLAB engine. Thus, unlike with the Simulink coupling, very few modifications have been made to the original MATLAB code for it to perform as needed in ModelCenter. Though no plug-in is available for STAR-CCM+, it is still relatively straightforward to create a component for the STAR-CCM+ model using QuickWrap, a general script-wrapping utility offered by ModelCenter. This utility makes use of a Java macro, which is automatically generated by STAR-CCM+, to run the melter simulation in batch mode. QuickWrap can also parse output files from STAR-CCM+ at user-specified delimiters to extract the data that is to be sent to MATLAB. In contrast to the Simulink coupling, linking the two models is almost trivial in ModelCenter, in which the creation of links between variables is easily accomplished through an intuitive, user-friendly GUI. In Simulink, linking the two models requires the use of tedious code that has yet to work. Additionally, ModelCenter, through the use of AnalysisServer, allows the various components in a model to run on different machines. This utility has been used in the melter model to run the computationally intensive STAR-CCM+ simulation on a high-performance computing cluster while the rest of the model runs on a desktop. In Simulink, this kind of machine flexibility would not be possible without significant programming efforts. Lastly, because of its user-friendly interface, the ModelCenter simulation is also much more maintainable than the Simulink model, as very few special skills are required to understand how to use ModelCenter.

The above outlined benefits of ModelCenter must be weighed against licensing costs, as our project has already incurred licensing costs for MATLAB and STAR-CCM+. The benefits of using ModelCenter, however, outweigh these costs. It is foreseeable that ModelCenter's optimization and trade study capabilities will allow us to optimize melting rate and glass quality for our melter model, thus leading to savings in money and time for waste vitrification operations. After using ModelCenter for this waste melter model, we recommend the use of this powerful tool in modeling other multi-component systems.

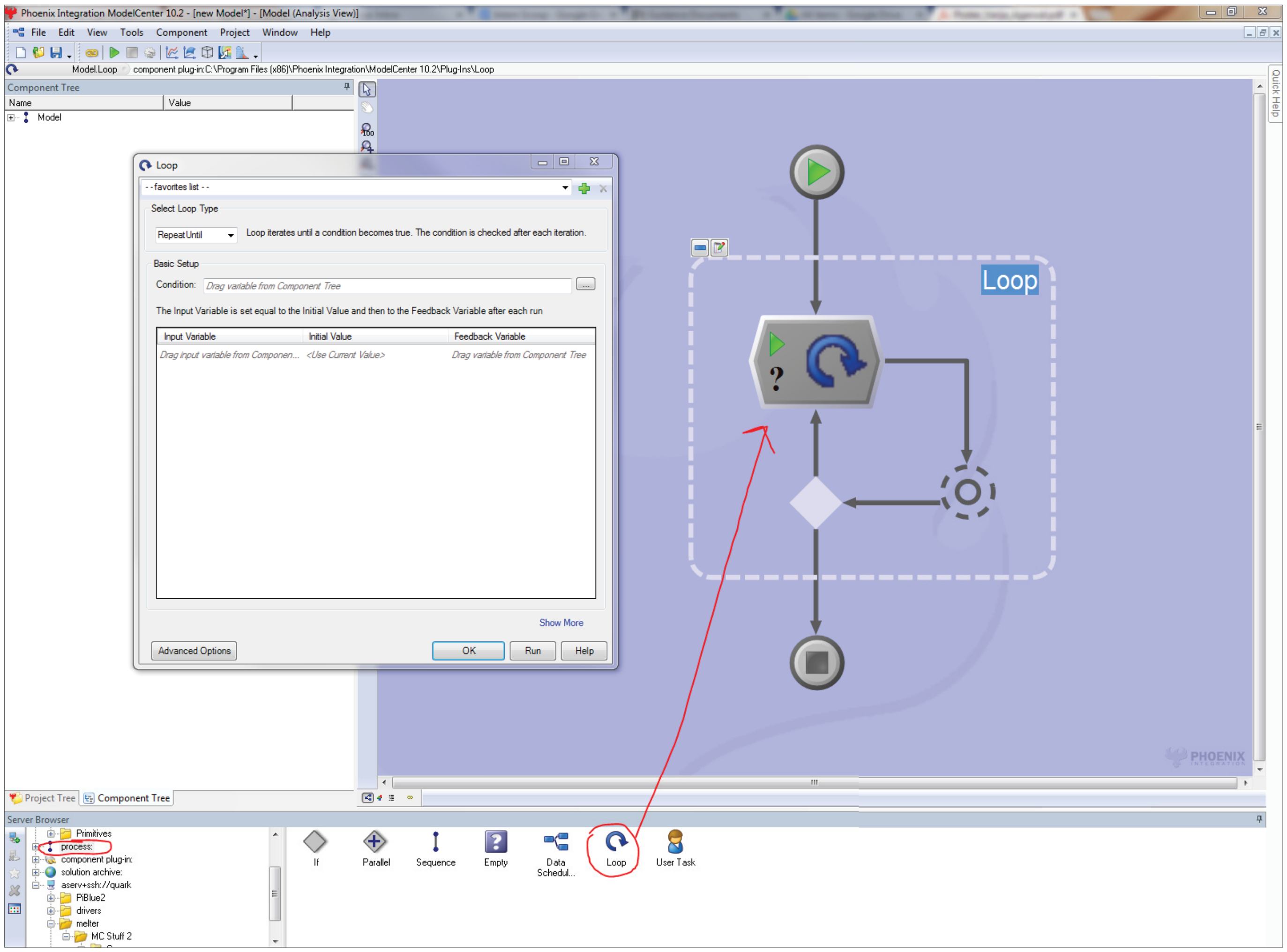
2. STARTING A NEW MODEL

When a new model is created in ModelCenter, it will prompt you to select from either a process flow or a data flow. For the melter project, we will use a process flow, as it is easier to implement a circular model using a loop in the process flow.


Phoenix Integration ModelCenter 10.2
File
Edit
View
Tools
Component
Project
Window
Help
Component Tree
Name
Value
What type of model would you like to create?
Use this model type to automate common processes
Use this model type to build multi-disciplinary models
Process
Data
Don't ask me again
Help
To start
select

2.1 Creating a Loop

The loop component will be used to drive the coupled model. To create a loop component, select "process": from the Server Browser on the bottom left, and then select "Loop" from the bottom pane. Drag and drop the loop into the purple model screen.



Model.Loop component plug-in:C:\Program Files (x86)\Phoenix Integration\ModelCenter 10.2\Plug-Ins\Loop

- - favorites list - -

Select Loop Type

Repeat Until — Loop iterates until a condition becomes true. The condition is checked after each iteration.

Basic Setup

Condition: *Drag variable from Component Tree*

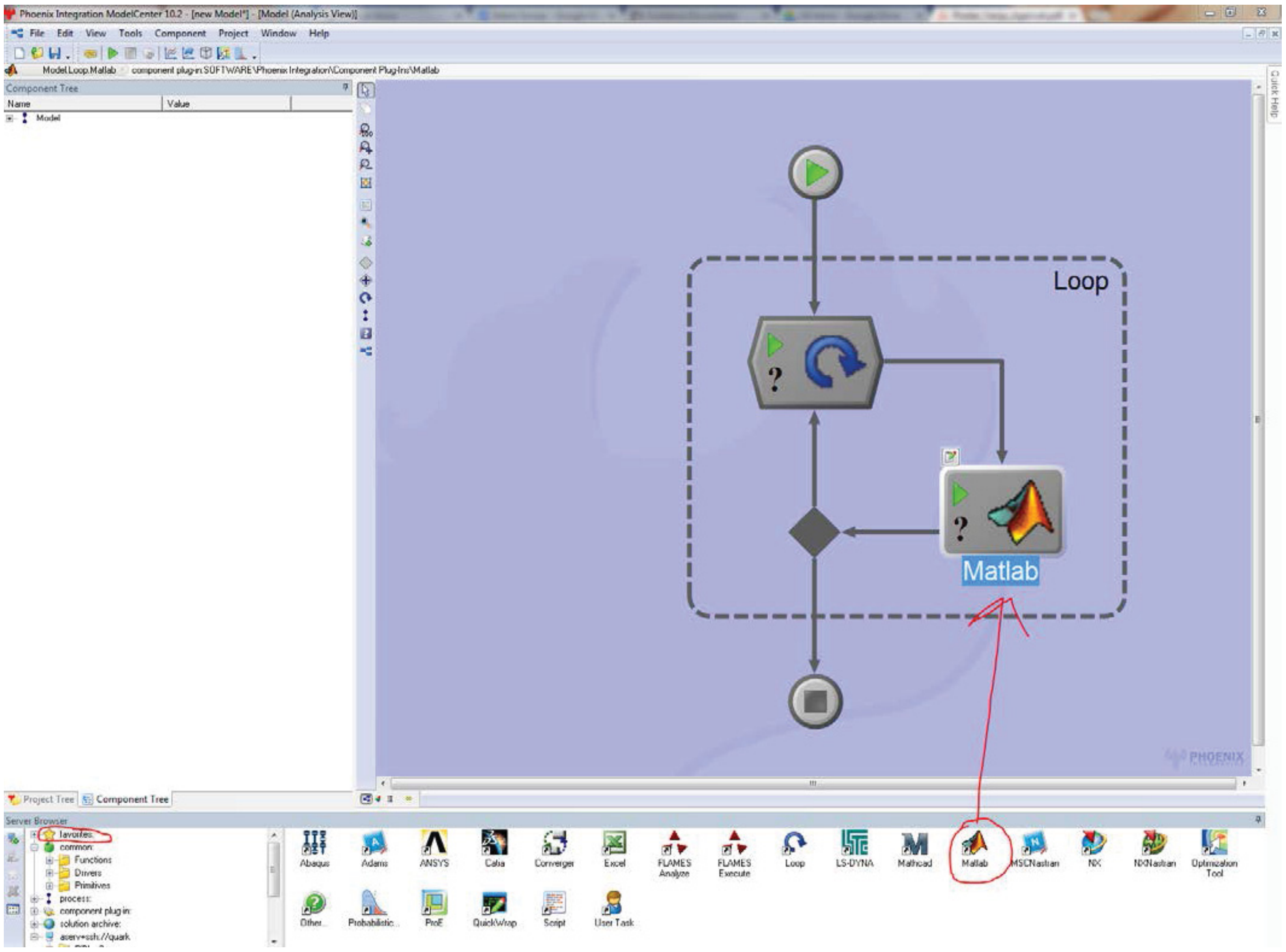
The Input Variable is set equal to the Initial Value and then to the Feedback Variable after each run

Input Variable	Initial Value	Feedback Variable
Drag input variable from Componen...	*<Use Current Value>*	*Drag variable from Component Tree*

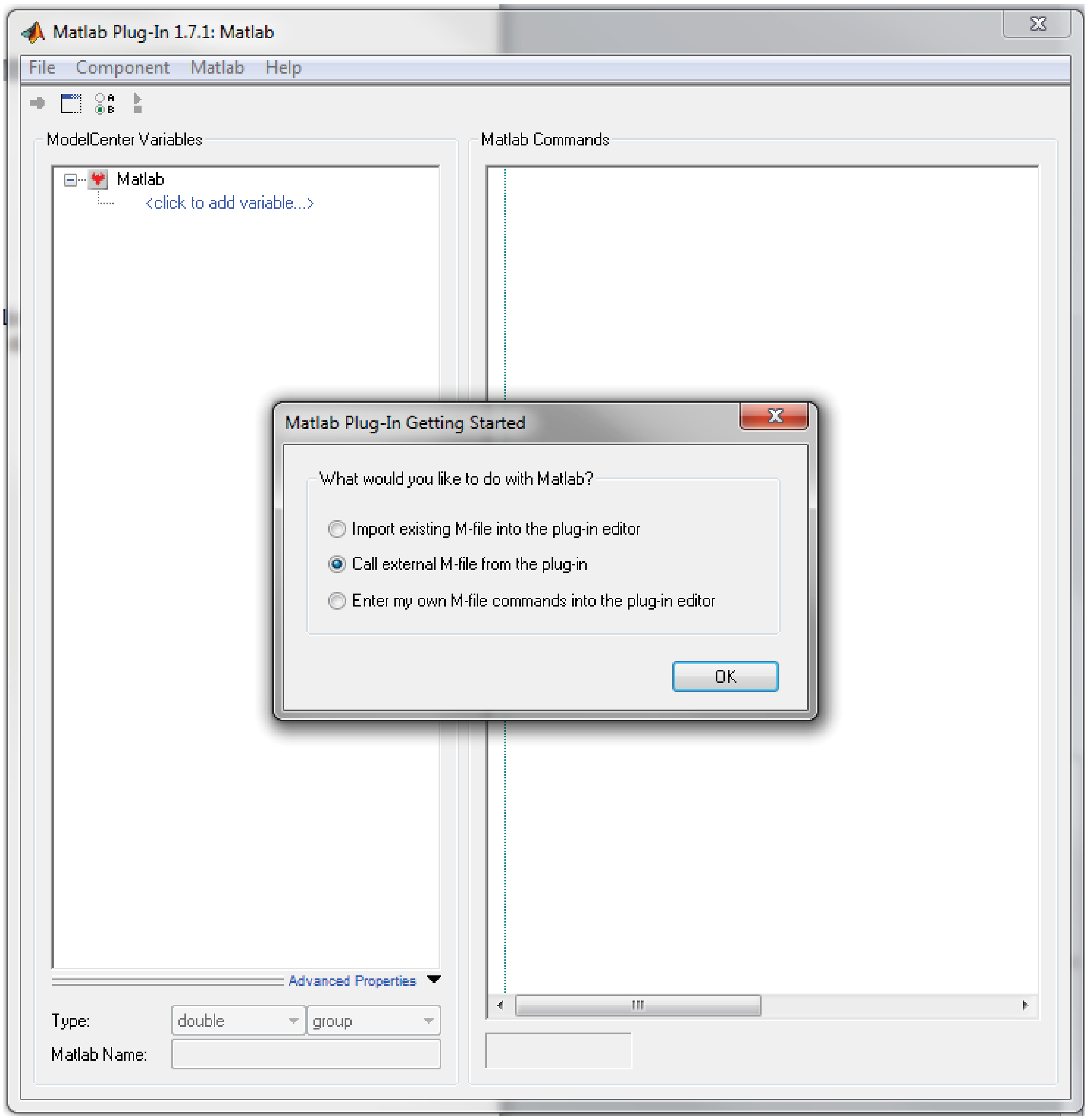
After you drag and drop the loop component into the model, a pop-up should appear that asks you to set the loop properties. Ignore this for now by either closing the window or pressing "OK", since loop properties will be set later in this tutorial.

2.2 Adding the MATLAB Component

To add the MATLAB model of the cold cap to the model, click on “favorites:” in the Server Browser on the bottom left of the screen, and drag and drop the MATLAB plug-in into the model.



As soon as you drag and drop the MATLAB component into the model, a window should pop-up, giving you three options for how you want to use MATLAB code. In our case, we are going to use the second option, “Call external M-file from the plug-in.”

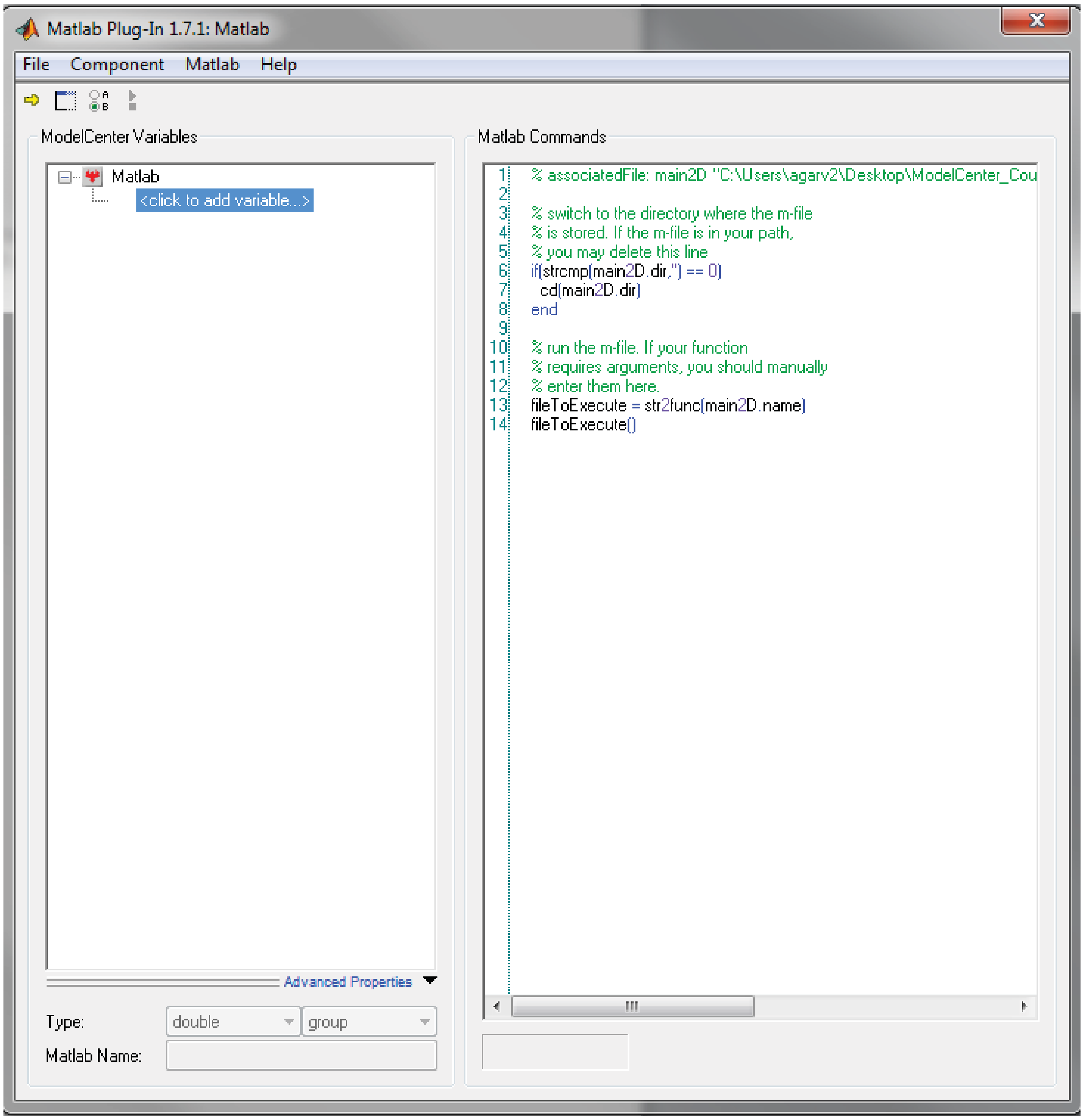

Matlab Plug-In 1.7.1: Matlab

Matlab Plug-In Getting Started

What would you like to do with Matlab?

- Import existing M-file into the plug-in editor
- [✓ Call external M-file from the plug-in]
- Enter my own M-file commands into the plug-in editor

You will then be asked to browse and select the MATLAB file that you want to run. For this project, navigate to the directory that contains all the cold cap model files, and select “main2D.m.” Once you have done so, a pop-up will show up that says “No variables were detected.” This is fine and can be ignored, since we will be setting variables manually. Code should now appear in the window, however, that links to your main2D.m file, as shown on the next page.



Type: double group

Matlab Name:

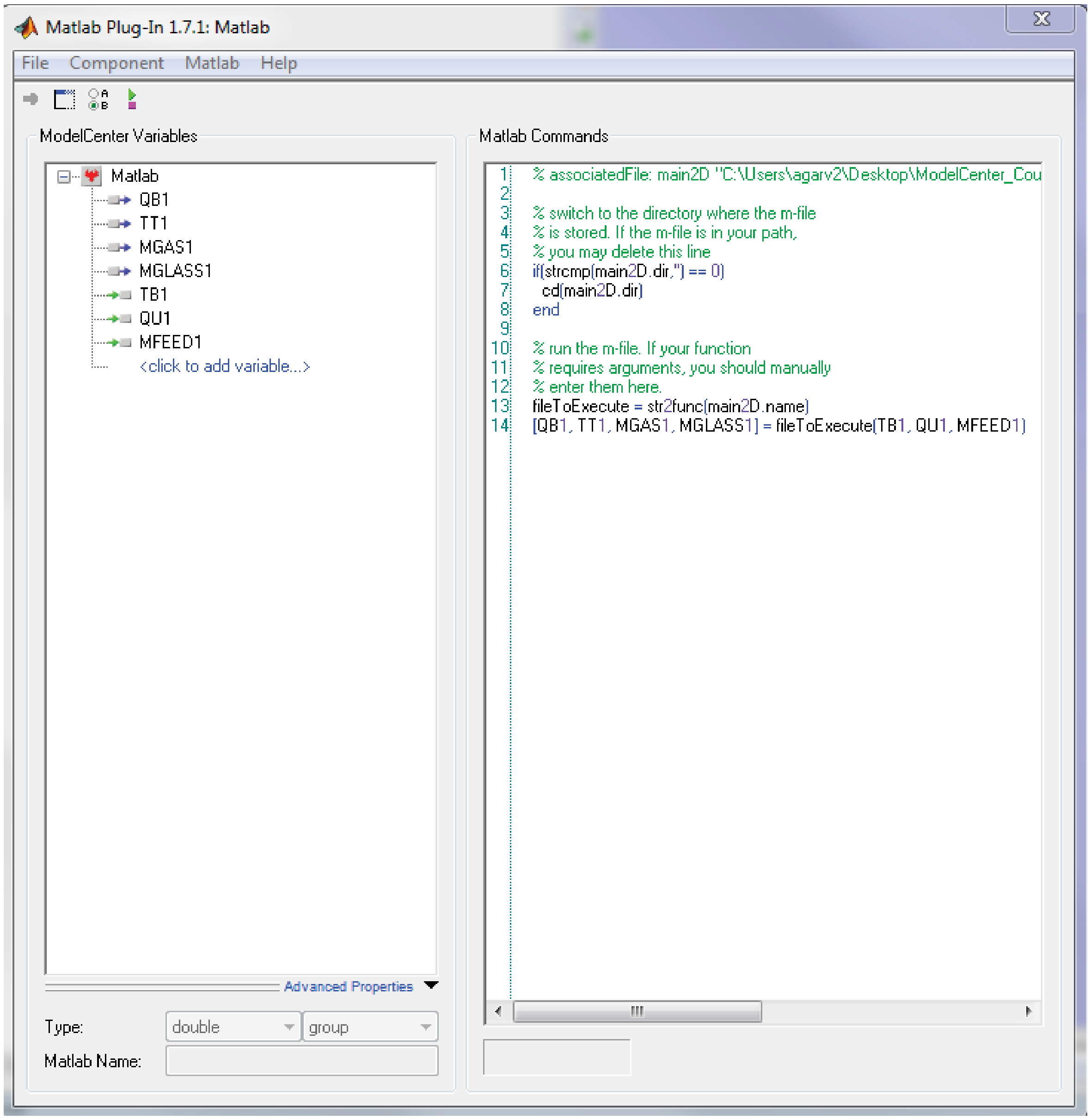
```
 1  % associatedFile: main2D "C:\Users\agarv2\Desktop\ModelCenter_Cou
 2
 3  % switch to the directory where the m-file
 4  % is stored. If the m-file is in your path,
 5  % you may delete this line
 6  if(strcmp(main2D.dir,'') == 0)
 7    cd(main2D.dir)
 8  end
 9
10  % run the m-file. If your function
11  % requires arguments, you should manually
12  % enter them here.
13  fileToExecute = str2func(main2D.name)
14  fileToExecute()
```

We now need to edit this code to reflect the inputs and outputs coming into and leaving from the MATLAB model. The MATLAB model will take in TB1, QU1 and MFEED1, and it will output QB1, TT1, MGAS1 and MGLASS1. Therefore, change the last line in the code (line 14) to be: `[QB1, TT1, MGAS1, MGLASS1] = fileToExecute(TB1, QU1, MFEED1)`

This is basically how a function is called in MATLAB; here you are calling the main2D function with the appropriate inputs and outputs. In the MATLAB file main2D.m, the function header must also have the same inputs and outputs.

Next, you must add all these variables in the left pane where it says <click to add variable…>. Specify each variable as an input or output, and also make sure the Type of each variable is what it needs to be (for our case it should be double for all these variables).

The new code and variables for the MATLAB ModelCenter component are shown below:



Matlab
QB1
TT1
MGAS1
MGLASS1
TB1
QU1
MFEED1
<click to add variable...>

```
% associatedFile: main2D "C:\Users\agarv2\Desktop\ModelCenter_Cou

% switch to the directory where the m-file
% is stored. If the m-file is in your path,
% you may delete this line
if(strcmp(main2D.dir,'') == 0)
  cd(main2D.dir)
end

% run the m-file. If your function
% requires arguments, you should manually
% enter them here.
fileToExecute = str2func(main2D.name)
[QB1, TT1, MGAS1, MGLASS1] = fileToExecute(TB1, QU1, MFEED1)
```

Type: double group

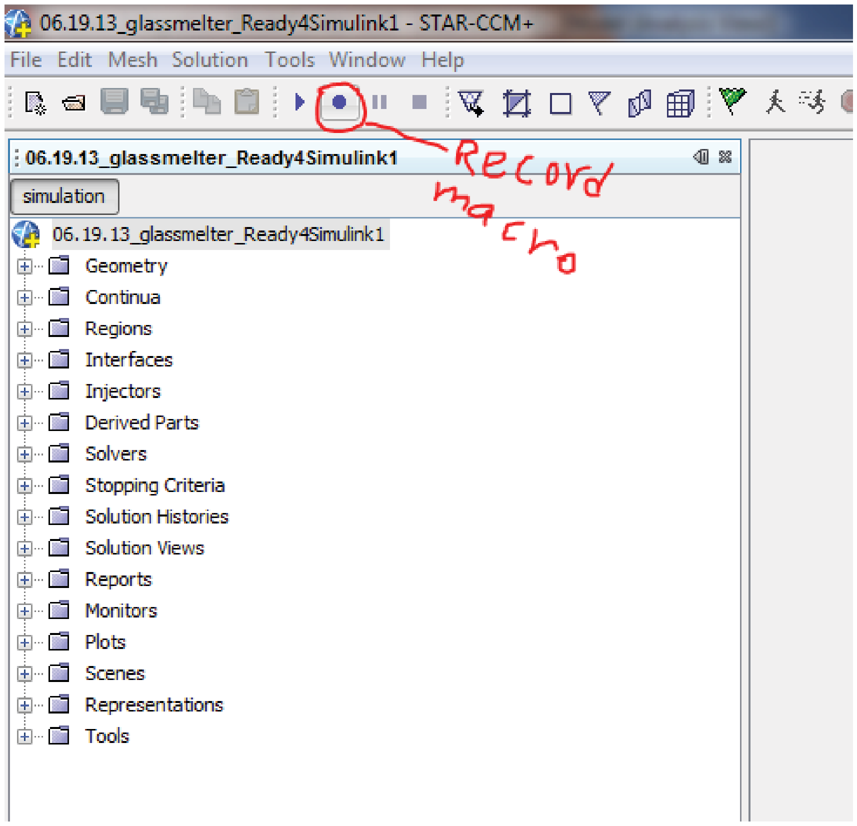
Matlab Name:

Once you are done making all the above changes, either press Control+S or click on the yellow arrow in the upper left corner to save changes to your component. Whenever you make any changes to any component, always remember to save!

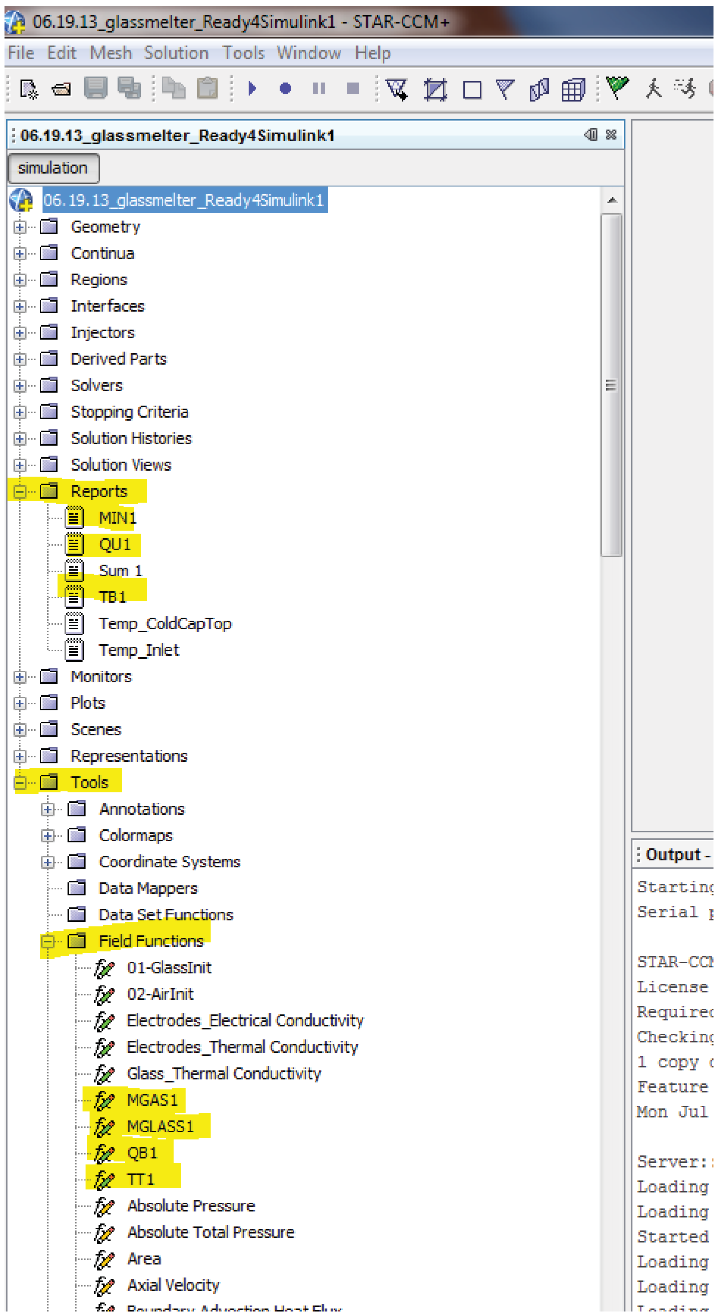
We are done with the MATLAB component, and the component window can now be closed.

2.3 Adding the STAR-CCM+ Component

Because a direct plug-in is not available for STAR-CCM+ as it is for MATLAB, the general script-wrapping utility, QuickWrap, will be used to create a component for the STAR-CCM+ melter model. This component will run STAR-CCM+ in batch mode (from a command window/terminal) by using the command "`starccm+ -batch <macro file name> <simulation file name>`". Therefore, a Java macro is needed from the STAR-CCM+ simulation. This macro will contain all the code needed to run the simulation. A macro can be created in STAR-CCM+ by first loading the simulation that you will be running (it's okay if you think you might use a different simulation in the future – the macro should, for the most part, still be the same) and then clicking on the blue circle button in the upper left, as shown below.



Once you click on the Record Macro button, you will be prompted to select a file to save the macro to. In this case, it is best to create a new file (let's call it "macro.java"). Once you select the file, recording has begun. Do not click around too much or change anything immediately, as whatever you do from now on in the simulation will be recorded as code in your macro file. Note that for our coupled model, STAR-CCM+ needs to have some inputs and outputs. The inputs will be MGAS1, MGLASS1, QB1 and TT1, and these should all be defined as field functions in the STAR simulation. The outputs will be QU1, TB1 and MIN1 (which is the same as MFEED1 later on in ModelCenter), and these should be defined as reports in the STAR simulation. The field and report setup is shown in the picture on the next page.



Once fields and reports are set in the STAR simulation, the following things need to be done when the macro begins recording:

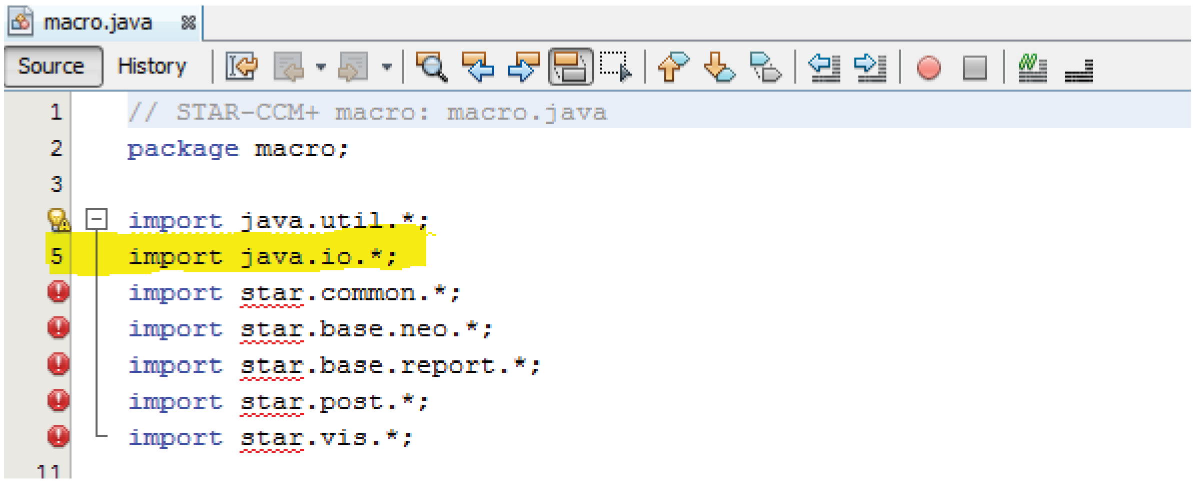
1. Click on each field function (MGAS1, MGLASS1, QB1, TT1), and set its value to something. The value can be set to anything, as we just want to get the macro to give us the general code for setting a field to a certain value).
2. Under Stopping Criteria, set the maximum physical time to reflect whatever is needed. Currently, in the macro used for the coupling, this is set to 0.01 seconds.
3. Run the simulation. You do not have to run the simulation for its entire length; you can stop the simulation as soon as you want, since all we really care about is to get a line of code in our macro that represents the running of the simulation.

4. Right-click on each report and select “Run Report.”
5. Open up whatever scenes you would like to export pictures of. Right-click on the scene, select “Hardcopy”, and save the picture as a .png file somewhere. I will be saving scenes for temperature, velocity streamlines, volume fractions of air, bubbles, glass and slurry and also residuals. I will be saving these in a folder called “STAR Pics.”

Now you may stop recording the macro by clicking on the blue square button that is in the same area as the blue circle button you used to start recording the macro.

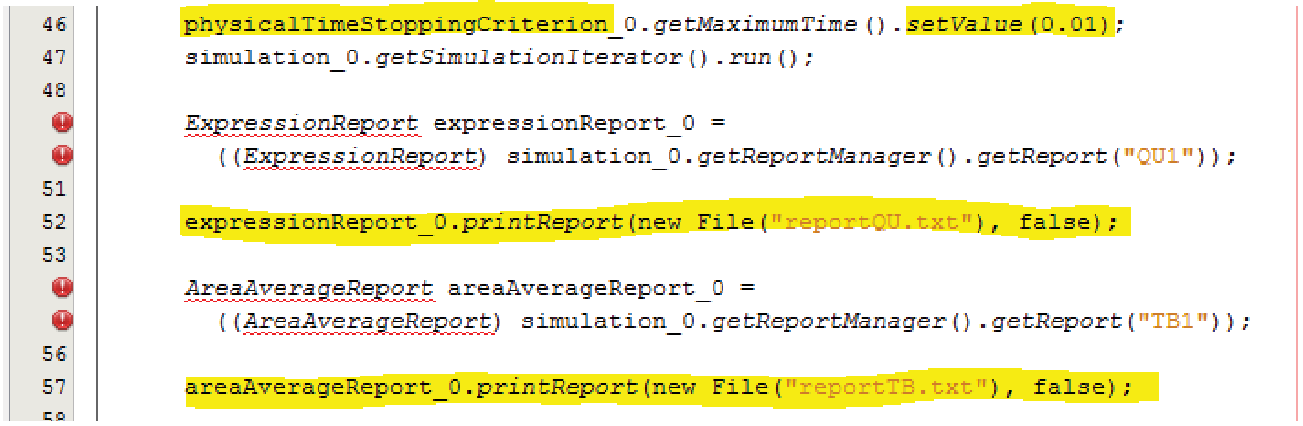
If you open “macro.java” (or view it in STAR-CCM+, as it should be displaying that file on another tab in the Output window), you will now see that there is code in the file for every step you performed while you were recording the macro. If someone were to play this macro in the STAR simulation used in the coupling, all the steps performed while recording will be executed automatically. The macro is what the ModelCenter model will be running, so now you know exactly what will be happening every time the STAR component in ModelCenter runs.

Now we need to make some minor changes to the macro. You can use a text editor (WordPad or NotePad++, for example) or a Java integrated development environment (like NetBeans or Eclipse) to edit a Java file. In this case, at the top of the file, with all the import statements, an “import java.io.*;” line needs to be added because we will be creating output files using the File class in the java.io library.



```
// STAR-CCM+ macro: macro.java
package macro;

import java.util.*;
import java.io.*;
import star.common.*;
import star.base.neo.*;
import star.base.report.*;
import star.post.*;
import star.vis.*;
```

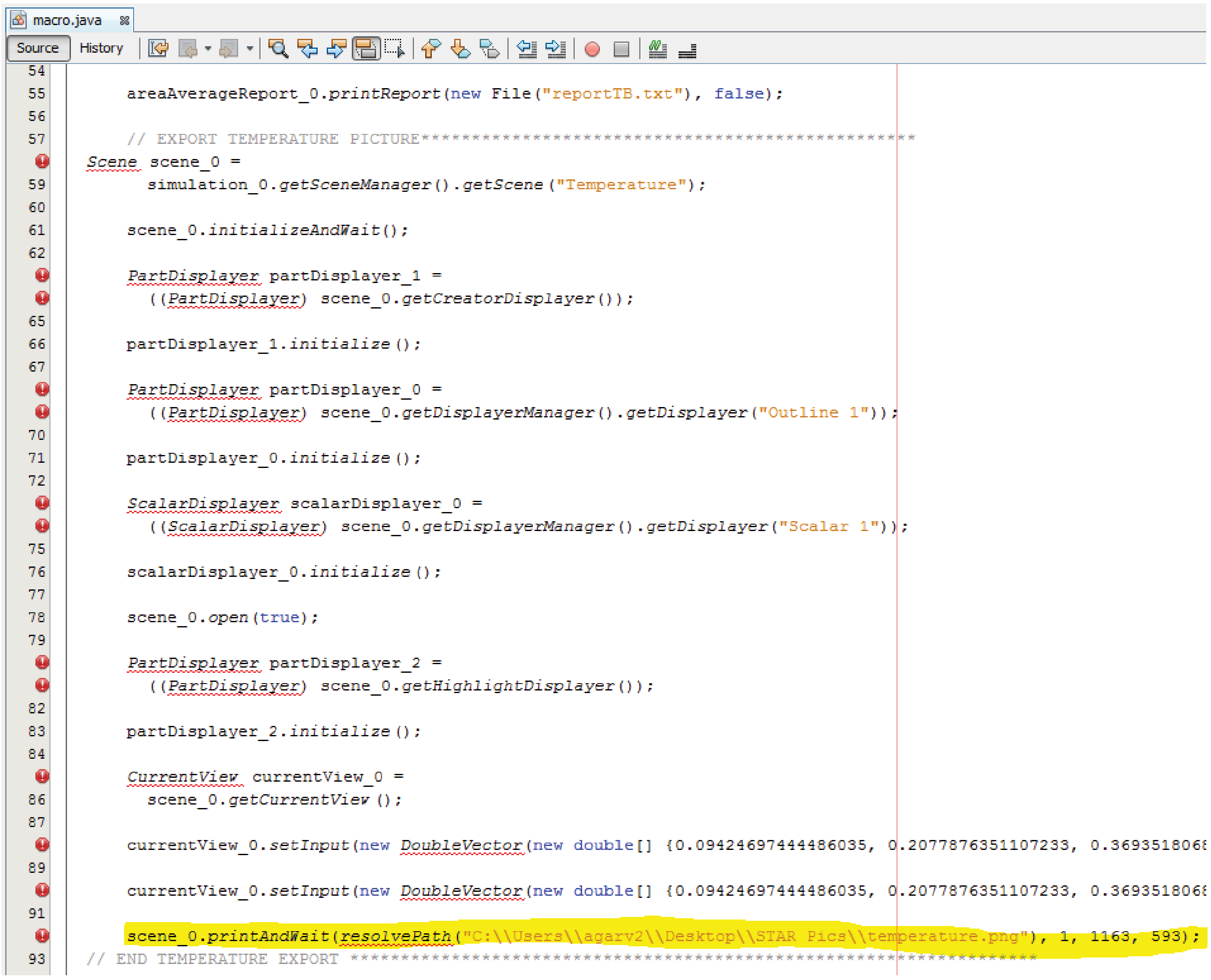
Next, towards the bottom of the file, the printReport statements need to be altered to print to a file, as can be seen in the picture. The “false” is to ensure that report files should not be appended to; that is, every report should clear out whatever was in the file previously.



```
46          physicalTimeStoppingCriterion_0.getMaximumTime().setValue(0.01);
47          simulation_0.getSimulationIterator().run();
48
            ExpressionReport expressionReport_0 =
              ((ExpressionReport) simulation_0.getReportManager().getReport("QU1"));
51
52          expressionReport_0.printReport(new File("reportQU.txt"), false);
53
            AreaAverageReport areaAverageReport_0 =
              ((AreaAverageReport) simulation_0.getReportManager().getReport("TB1"));
56
57          areaAverageReport_0.printReport(new File("reportTB.txt"), false);
```

Notice that if you ever need to change how long the simulation runs, you can simply change that value in this macro in line 64 as seen in the picture above. Right now, the simulation is set to run for 0.01 seconds (physical time), but that value can be easily changed to any other number.

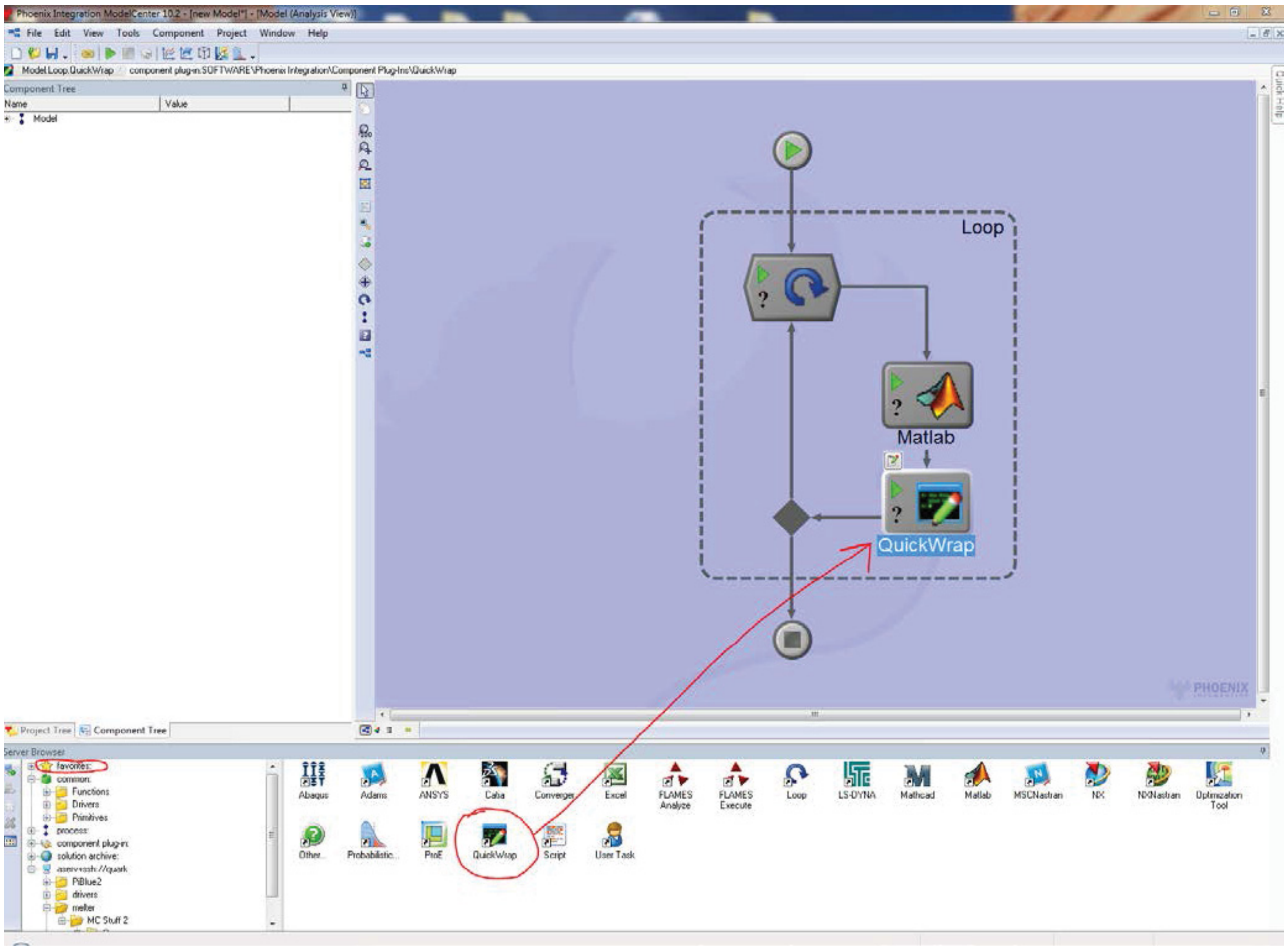
Next, we need to take a look at the code that exports scenes as pictures. For the purposes of this manual, let's focus on the macro code that deals with the temperature scene. The last line of code in the section shown below is what we are interested in: this line saves the scene as a picture file to a certain path. Make sure that the path and filename are what you want. I want to save my temperature picture to a file called "temperature.png" in the "Star Pics" folder on my desktop; therefore, the line of code is correct. Make sure the filename and path are correct for all the scenes you are exporting.



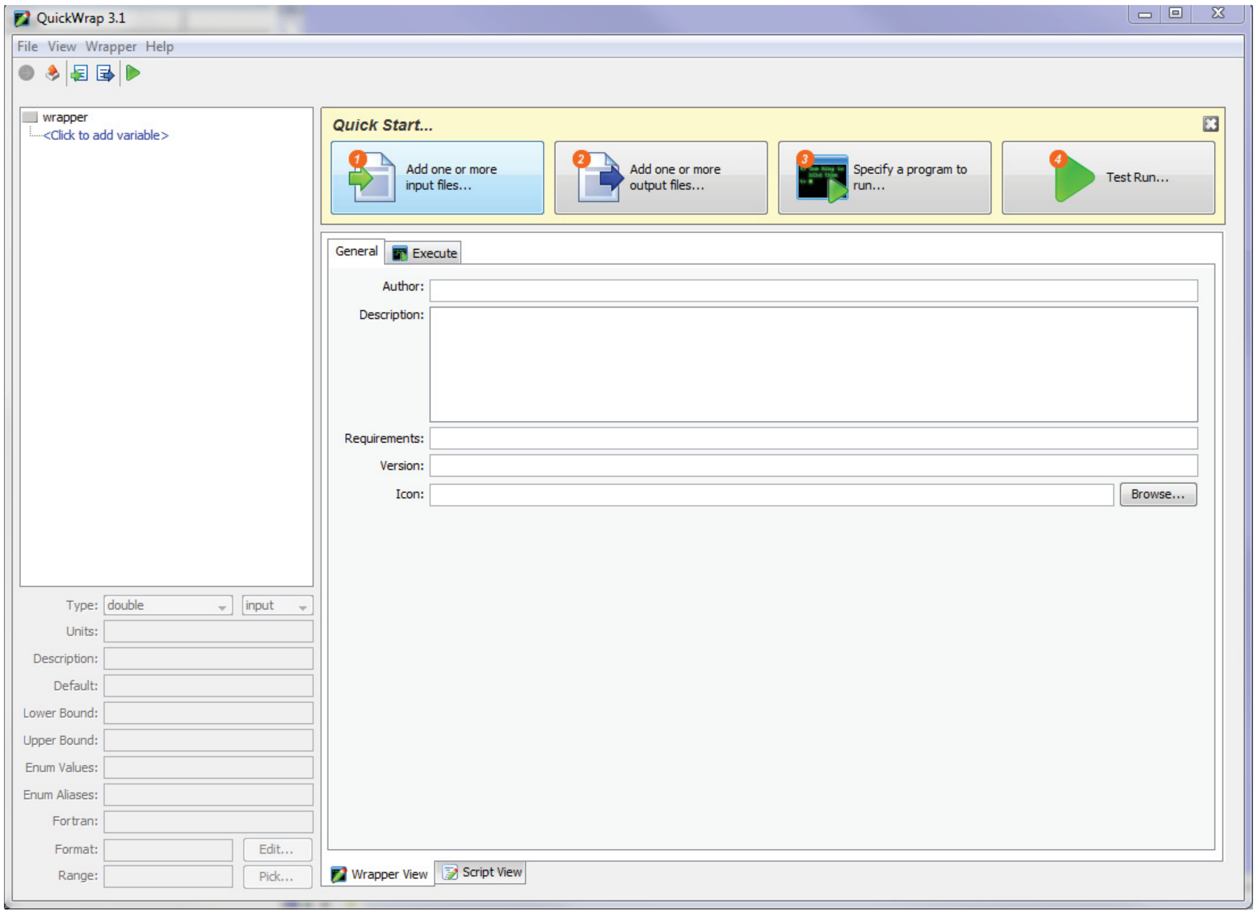
```
      areaAverageReport_0.printReport(new File("reportTB.txt"), false);

      // EXPORT TEMPERATURE PICTURE**************************************************
  Scene scene_0 =
        simulation_0.getSceneManager().getScene("Temperature");

      scene_0.initializeAndWait();

      PartDisplayer partDisplayer_1 =
        ((PartDisplayer) scene_0.getCreatorDisplayer());

      partDisplayer_1.initialize();

      PartDisplayer partDisplayer_0 =
        ((PartDisplayer) scene_0.getDisplayerManager().getDisplayer("Outline 1"));

      partDisplayer_0.initialize();

      ScalarDisplayer scalarDisplayer_0 =
        ((ScalarDisplayer) scene_0.getDisplayerManager().getDisplayer("Scalar 1"));

      scalarDisplayer_0.initialize();

      scene_0.open(true);

      PartDisplayer partDisplayer_2 =
        ((PartDisplayer) scene_0.getHighlightDisplayer());

      partDisplayer_2.initialize();

      CurrentView currentView_0 =
        scene_0.getCurrentView();

      currentView_0.setInput(new DoubleVector(new double[] {0.09424697444486035, 0.2077876351107233, 0.3693518068

      currentView_0.setInput(new DoubleVector(new double[] {0.09424697444486035, 0.2077876351107233, 0.3693518068

      scene_0.printAndWait(resolvePath("C:\\Users\\agarv2\\Desktop\\STAR Pics\\temperature.png"), 1, 1163, 593);
  // END TEMPERATURE EXPORT ************************************************************************
```

We are now done editing the macro, and we can start setting up the STAR-CCM+ component in ModelCenter using QuickWrap.

From the same pane that had the MATLAB plug-in, select QuickWrap, and drag and drop it into the model.

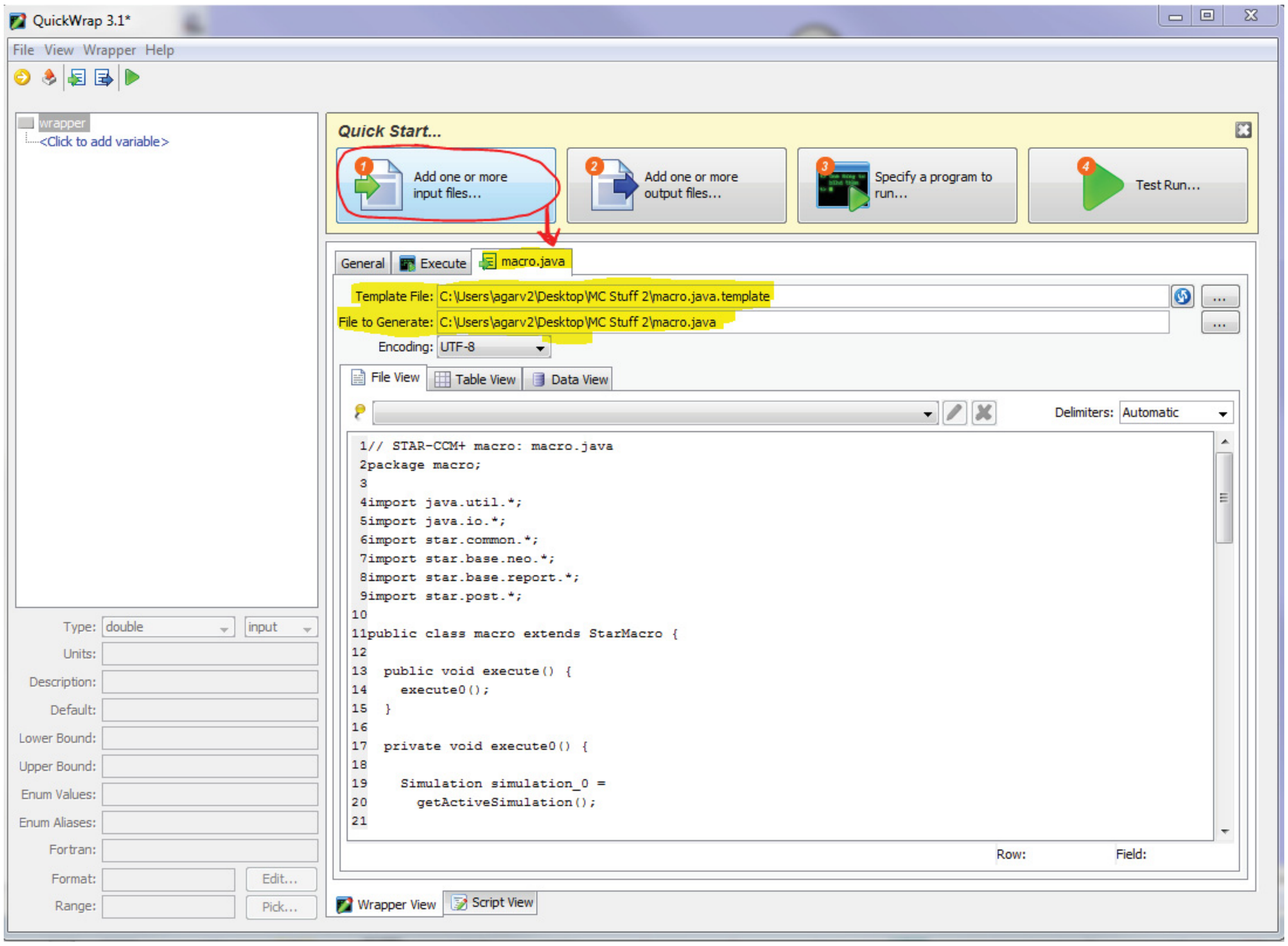


A component editor window should pop-up as soon as you drop the QuickWrap component into the model.



Before going further, create a new folder somewhere (I made one on my Desktop and called it MC Stuff 2), and add to it your "macro.java" file as well as your STAR simulation file). In this folder, make a copy of the "macro.java" file and name this copy "macro.java.template."

Once you are done with this, go back to the QuickWrap component editor and click on "Add one or more input files…." Select the macro.java file. It should ask you if you want to use the template file detected, and you should select "Yes." Notice that it now displays the macro code in the window, as shown on the next page. Unfortunately, ModelCenter does not allow you to edit the macro code in this window, so if you ever have to make any changes to the macro, you must use a text editor or IDE to edit the macro, and then you must reload it into the QuickWrap component. In that case, you would also have to re-create the "macro.java.template" file and reload that into the component as well.

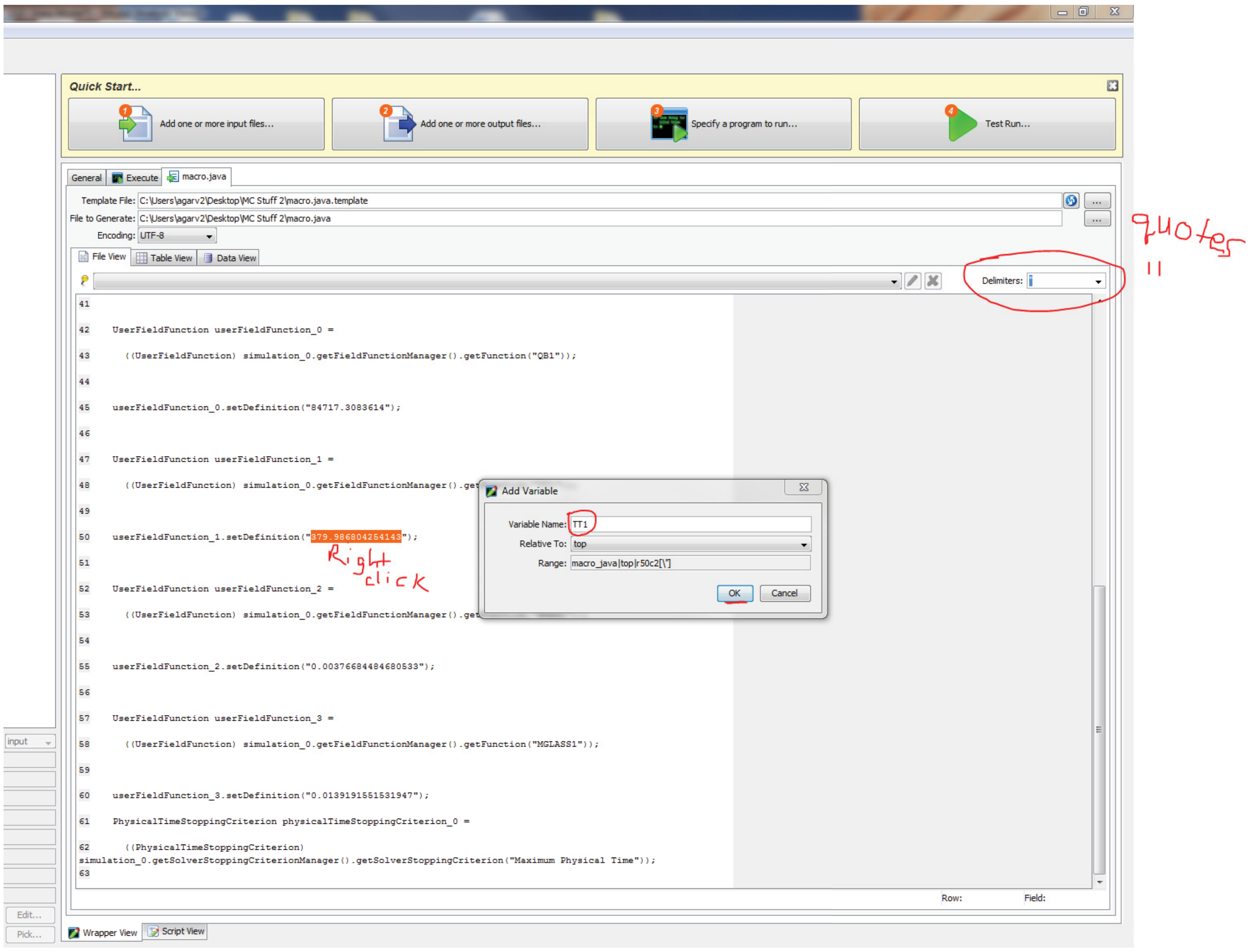


Template File: C:\Users\agarv2\Desktop\MC Stuff 2\macro.java.template
File to Generate: C:\Users\agarv2\Desktop\MC Stuff 2\macro.java
Encoding: UTF-8
Delimiters: Automatic

Type: double | input
Units:
Description:
Default:
Lower Bound:
Upper Bound:
Enum Values:
Enum Aliases:
Fortran:
Format:
Range:

```
// STAR-CCM+ macro: macro.java
package macro;

import java.util.*;
import java.io.*;
import star.common.*;
import star.base.neo.*;
import star.base.report.*;
import star.post.*;

public class macro extends StarMacro {

  public void execute() {
    execute0();
  }

  private void execute0() {

    Simulation simulation_0 =
      getActiveSimulation();

```

Scroll down to the lines of code that set values for field functions (such as userFieldFunction_0.setDefinition("42")). You should now specify the delimiter in the upper right hand corner just above the code window to be a quotation mark ("). Now if you hover your mouse over the number between the quotes in the setDefinition, you should be able to right-click on the highlighted number and click on "Add Variable". Add the appropriate variable as an input, make sure that the Type for each is specified to double, and specify a default value for your variable if you wish to do so.



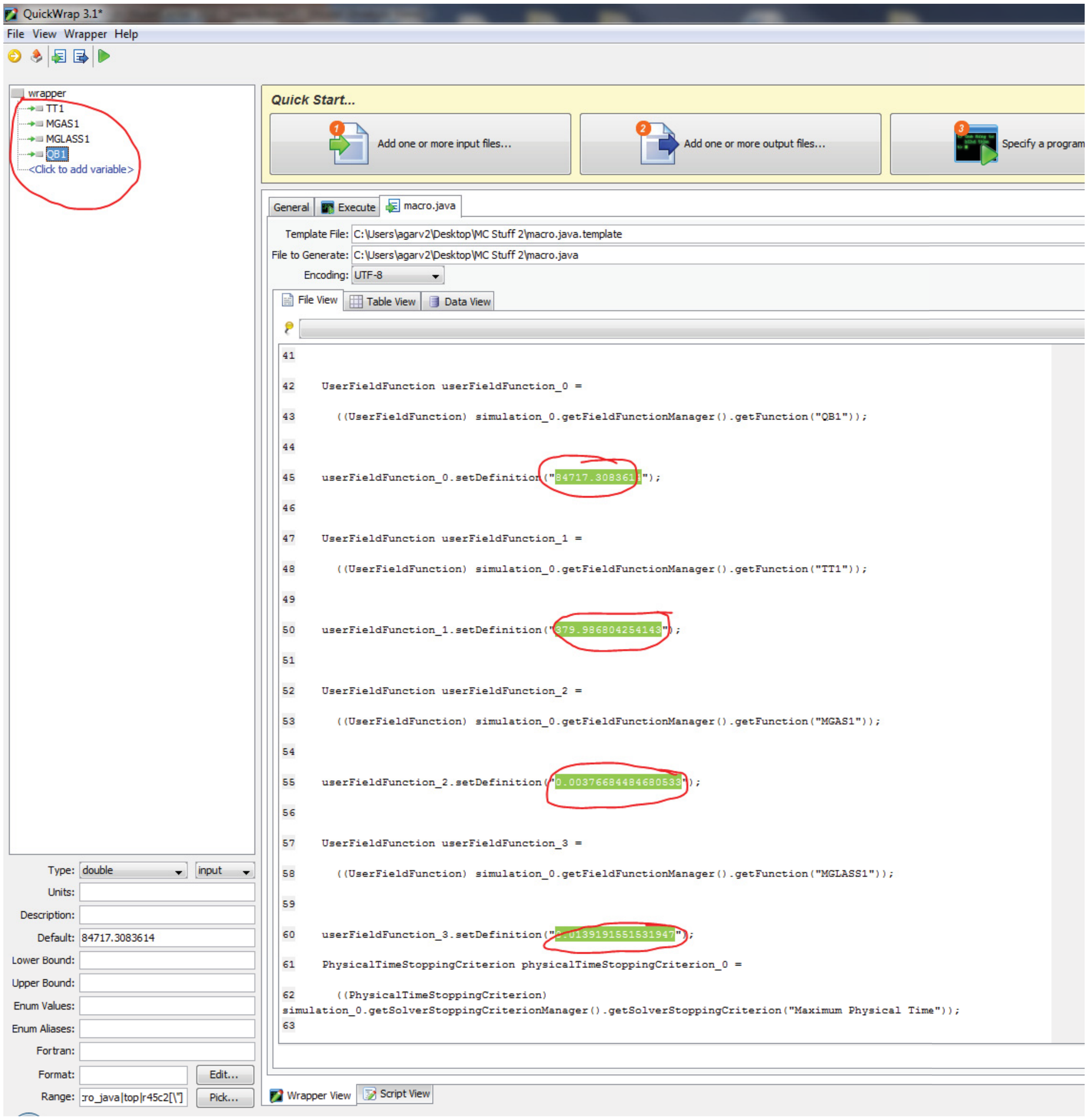
Template File: C:\Users\agarv2\Desktop\MC Stuff 2\macro.java.template
File to Generate: C:\Users\agarv2\Desktop\MC Stuff 2\macro.java
Encoding: UTF-8
Delimiters: "

```
UserFieldFunction userFieldFunction_0 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().getFunction("QB1"));


userFieldFunction_0.setDefinition("84717.3083614");


UserFieldFunction userFieldFunction_1 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().get


userFieldFunction_1.setDefinition("379.986804254143");


UserFieldFunction userFieldFunction_2 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().get


userFieldFunction_2.setDefinition("0.00376684484680533");


UserFieldFunction userFieldFunction_3 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().getFunction("MGLASS1"));


userFieldFunction_3.setDefinition("0.0139191551531947");

PhysicalTimeStoppingCriterion physicalTimeStoppingCriterion_0 =

  ((PhysicalTimeStoppingCriterion)
simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion("Maximum Physical Time"));
```

Add Variable

Variable Name: TT1
Relative To: top
Range: macro_java|top|r50c2[\"]
OK Cancel

Once have added the input variables, they should all appear on the left-hand pane, and it should be clear that they are linked to the text highlighted in green.



Template File: C:\Users\agarv2\Desktop\MC Stuff 2\macro.java.template
File to Generate: C:\Users\agarv2\Desktop\MC Stuff 2\macro.java
Encoding: UTF-8

```
UserFieldFunction userFieldFunction_0 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().getFunction("QB1"));

userFieldFunction_0.setDefinition("84717.3083614");

UserFieldFunction userFieldFunction_1 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().getFunction("TT1"));

userFieldFunction_1.setDefinition("379.986804254143");

UserFieldFunction userFieldFunction_2 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().getFunction("MGAS1"));

userFieldFunction_2.setDefinition("0.00376684484680533");

UserFieldFunction userFieldFunction_3 =

  ((UserFieldFunction) simulation_0.getFieldFunctionManager().getFunction("MGLASS1"));

userFieldFunction_3.setDefinition("0.0139191551531947");
PhysicalTimeStoppingCriterion physicalTimeStoppingCriterion_0 =

  ((PhysicalTimeStoppingCriterion)
simulation_0.getSolverStoppingCriterionManager().getSolverStoppingCriterion("Maximum Physical Time"));
```

Type: double input
Units:
Description:
Default: 84717.3083614
Lower Bound:
Upper Bound:
Enum Values:
Enum Aliases:
Fortran:
Format:
Range: :ro_java|top|r45c2[\"]

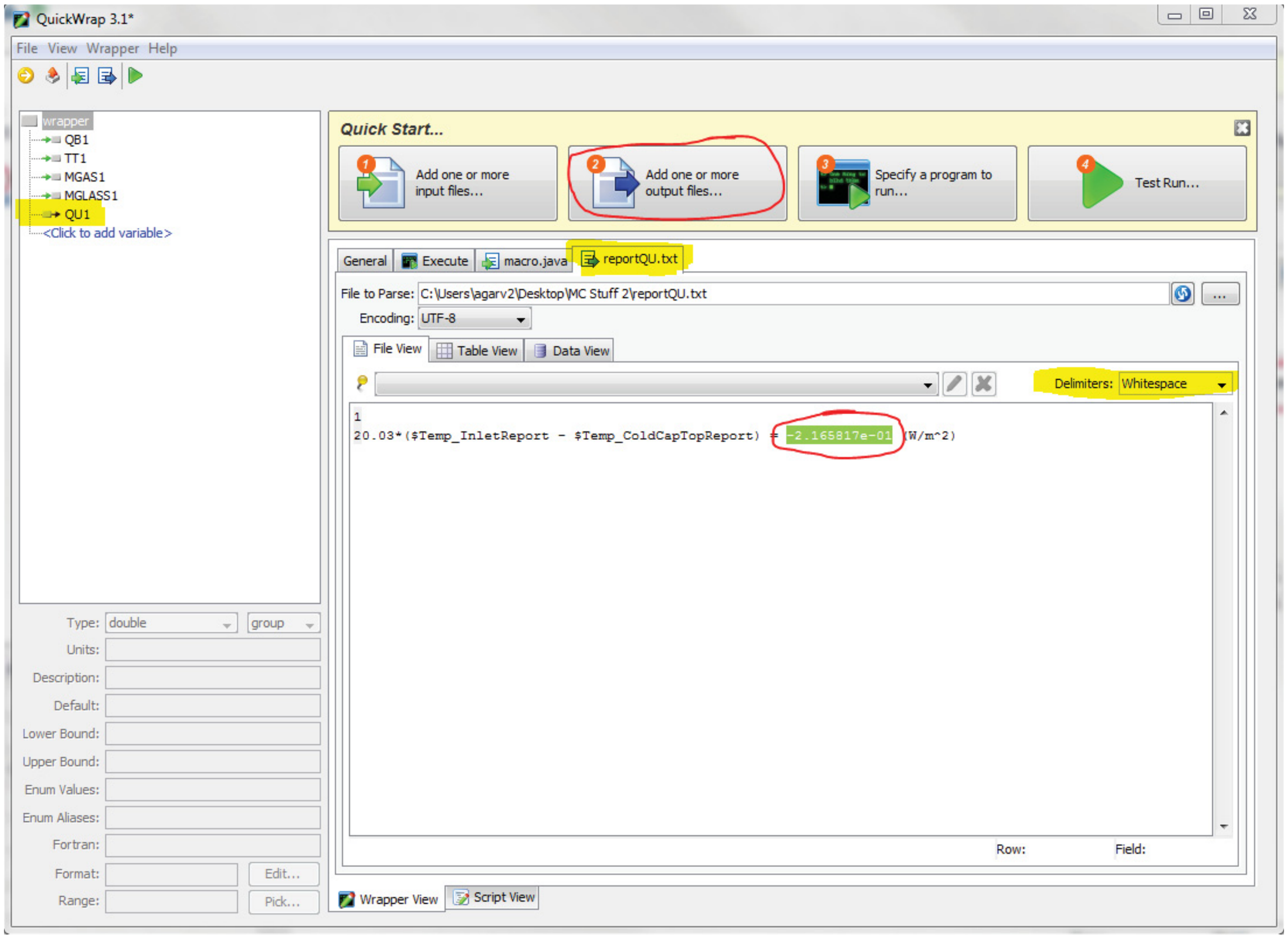
Now that we are done specifying the inputs, we will specify the output fields (QU1 and TB1). If you do not have copies of report files already available, the first thing you will need to do is open your command prompt/terminal and run your STAR simulation in batch mode using the macro. Type in the following once you are in the terminal, replacing the <FULL PATH TO MACRO> with the absolute pathname to "macro.java" and replacing the <FULL PATH TO SIM FILE> with the absolute pathname to your .sim file:

```
starccm+ -batch <FULL PATH TO MACRO>  <FULL PATH TO SIM FILE>
```

Your STAR-CCM+ model should now be running. Once it runs, you should have two output files, "reportQU.txt" and "reportTB.txt" in the current directory. You need to then add these two files to the folder that contains your macro file, macro template, and STAR simulation (in my case, this folder is MC Stuff 2).

Once this is done, go back to the QuickWrap component editor and click on “Add one or more output files….” Select “reportQU.txt” from its folder, and when the “Choose File Type” dialog comes up, the “Parse file as” option should be set to ASCII. Close the “Auto Import Variables” dialog that comes up – it’s unnecessary. The contents of “reportQU.txt” should now be displayed in the File View box.

Change the delimiter to “Whitespace,” and then select the number that is your output. Right click on the number, and select “Add variable.” In this case, add the variable as QU1. Set the Type to be double, and change the default value if you wish to do so. You should end up with something like this:



File to Parse: C:\Users\agarv2\Desktop\MC Stuff 2\reportQU.txt

Encoding: UTF-8

Delimiters: Whitespace

```
1
20.03*($Temp_InletReport - $Temp_ColdCapTopReport) = -2.165817e-01 (W/m^2)
```

Type: double | group

Units:

Description:

Default:

Lower Bound:

Upper Bound:

Enum Values:

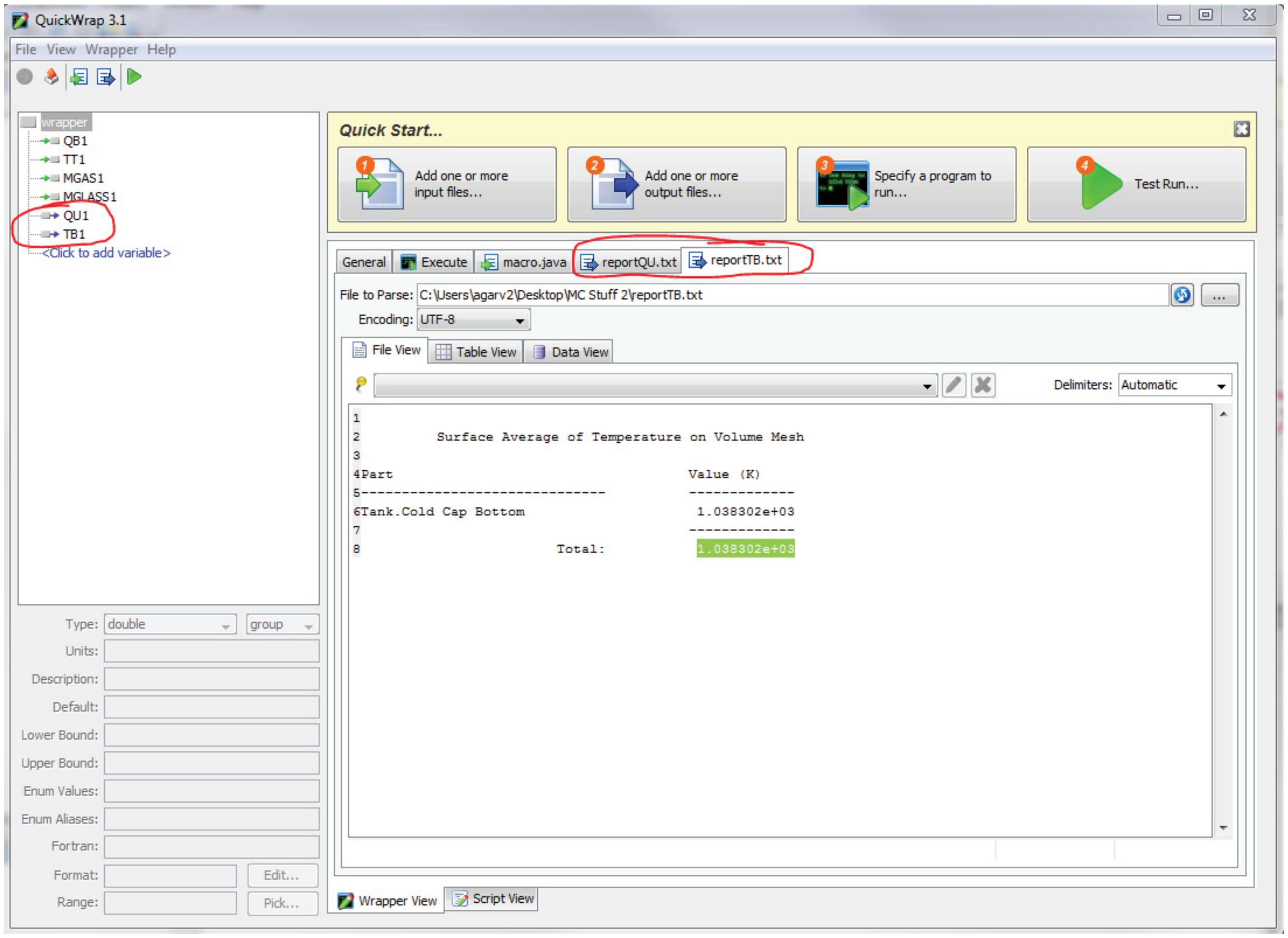
Enum Aliases:

Fortran:

Format:

Range:

Now do the same for reportTB.txt. Make sure you are saving all this as you go along. You should now have all your output variables set up for the STAR component, as shown below.

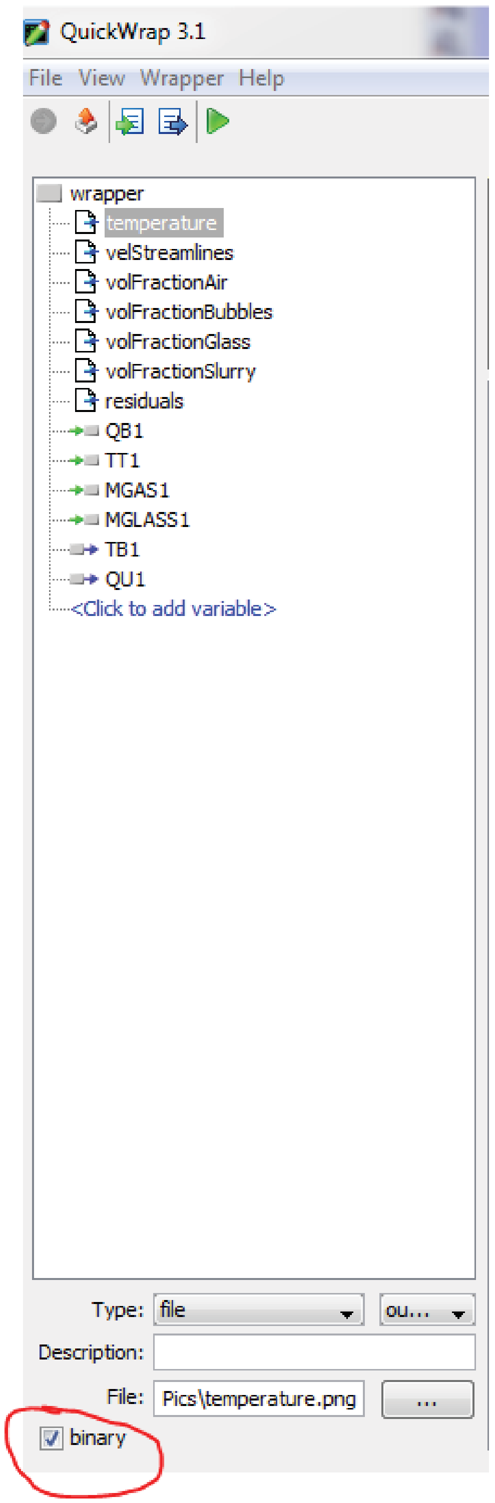


File to Parse: C:\Users\agarv2\Desktop\MC Stuff 2\reportTB.txt
Encoding: UTF-8
Delimiters: Automatic

```
1
2         Surface Average of Temperature on Volume Mesh
3
4Part                                  Value (K)
5------------------------------        -------------
6Tank.Cold Cap Bottom                   1.038302e+03
7                                       -------------
8                         Total:        1.038302e+03
```

Type: double group
Units:
Description:
Default:
Lower Bound:
Upper Bound:
Enum Values:
Enum Aliases:
Fortran:
Format: Edit...
Range: Pick...

The last few variables that we will be setting up are for the code that exports scenes as pictures. Right click on <click to add variable…>, and add the following variables as outputs: temperature, velStreamlines, volFractionAir, volFractionBubbles, volFractionGlass, volFractionSlurry and residuals. The "Type" of each of these variables should be changed to "file." When you change the type to "file," a prompt will come up, asking you to select the file it refers to. In this case, I navigated to the "STAR Pics" folder and selected the appropriate picture for each variable. Be sure to check the box labeled "binary" for each file variable.



Type: file ou...
Description:
File: Pics\temperature.png
[✓] binary

You should now have all of the variables set up for the STAR component.

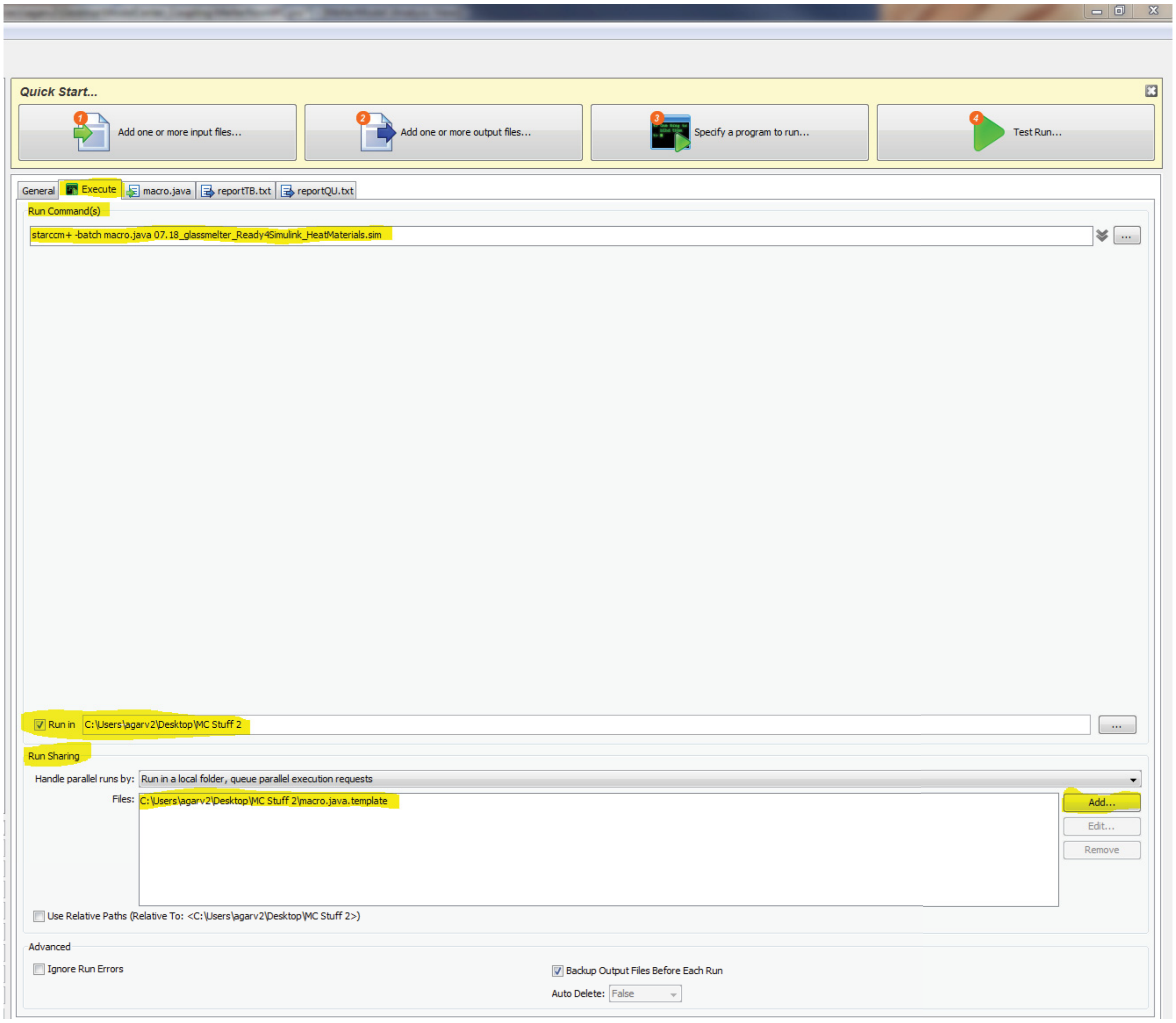
Now click on the “Execute” tab.

For Run Command(s), type in the command you used previously to run the simulation in the terminal, except this time, you do not need to specify the *absolute* paths to the macro and .sim files. This is because we will specify a directory to run the component out of. So for Run Command, type in the following, replacing the <SIM FILE NAME> with just the name of your simulation file:

```
starccm+ -batch macro.java <SIM FILE NAME>
```

Next, check the box next to “Run in” and select the folder that contains your macro, macro template, simulation file, and reports.

Under “Run sharing”, add your macro template file.



Run Command(s)

starccm+ -batch macro.java 07.18_glassmelter_Ready4Simulink_HeatMaterials.sim

[✓] Run in C:\Users\agarv2\Desktop\MC Stuff 2

Run Sharing

Handle parallel runs by: Run in a local folder, queue parallel execution requests

Files: C:\Users\agarv2\Desktop\MC Stuff 2\macro.java.template

[ ] Use Relative Paths (Relative To: <C:\Users\agarv2\Desktop\MC Stuff 2>)

Advanced

[ ] Ignore Run Errors

[✓] Backup Output Files Before Each Run

Auto Delete: False

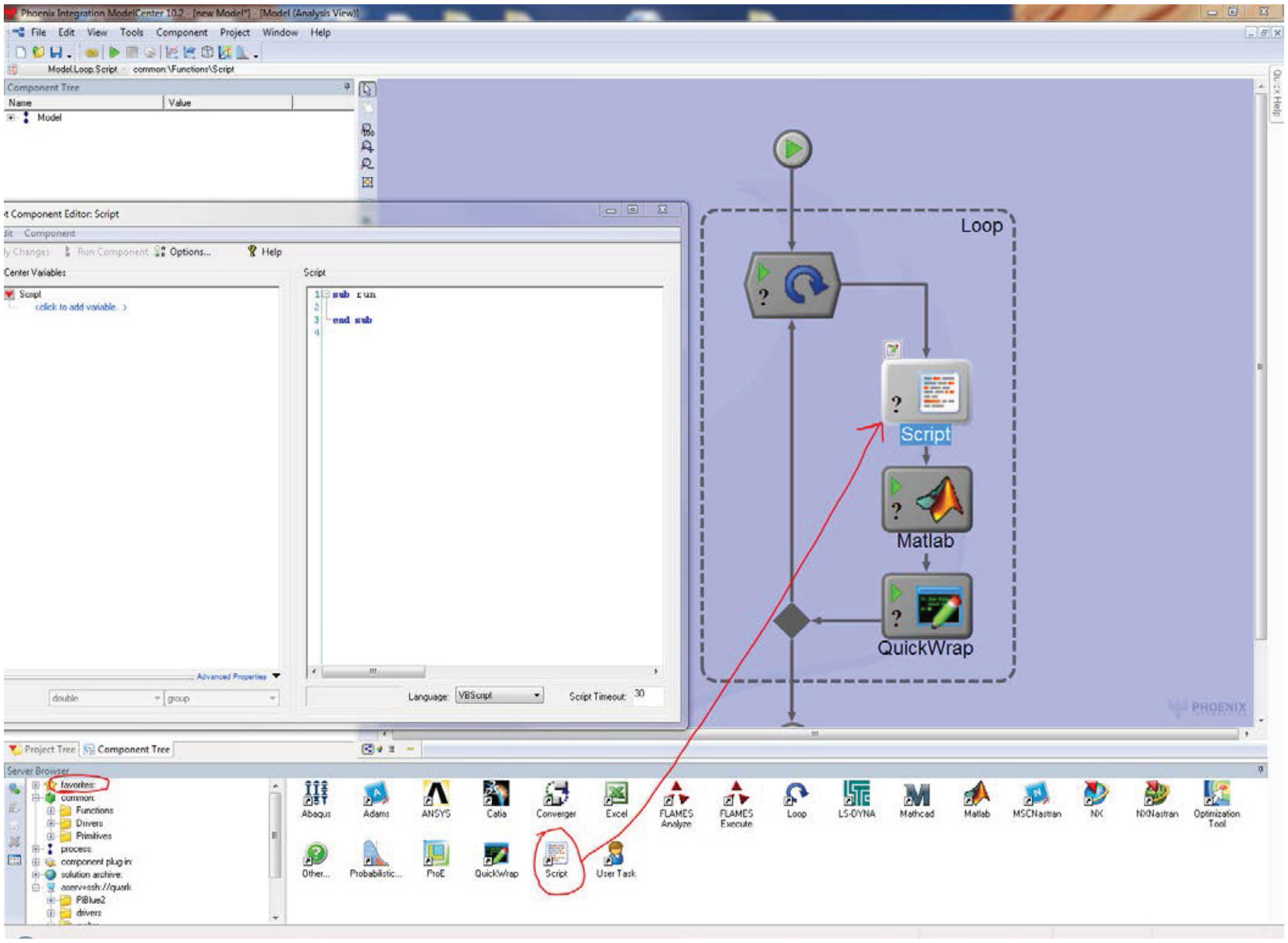
Now you can generate a wrapper file and save it in the same folder you have been working out of (MC Stuff 2). To generate a wrapper file, click on the “Export” button in the upper left hand corner (next to save button). This will create a “macro.scriptWrapper” file that you can save to your working directory. You shouldn’t need to do anything to this file to get it to work.

Save changes. We are now done with wrapping the STAR-CCM+ model. You can perform a test run on the component to see if it is working. If there is some sort of error, and it says that it was unable to run the starccm+ -batch command, there might be an issue with your license. In that case, try running the simulation in batch mode from the terminal as you did before and see if it gives you an error.

In the future, if the STAR simulation being used is changed, the “Run Command” will have to be changed to reflect the new name of the simulation file, and a new script wrapper will have to be generated for the HPC version of the model. Additionally, if any input or output variables are changed, those changes will obviously have to be reflected in the component variables.

2.4 Adding a Script Component to Output MFEED, Perform Bounds Checking and Control Iterations

Now we are going to add a component that will determine the value for the mass flux of slurry entering the cold cap, perform bounds checking on the data generated by the STAR-CCM+ model and control how many times the loop runs. We will now be using the "Script" option from the bottom pane. Drag and drop this into the model window.

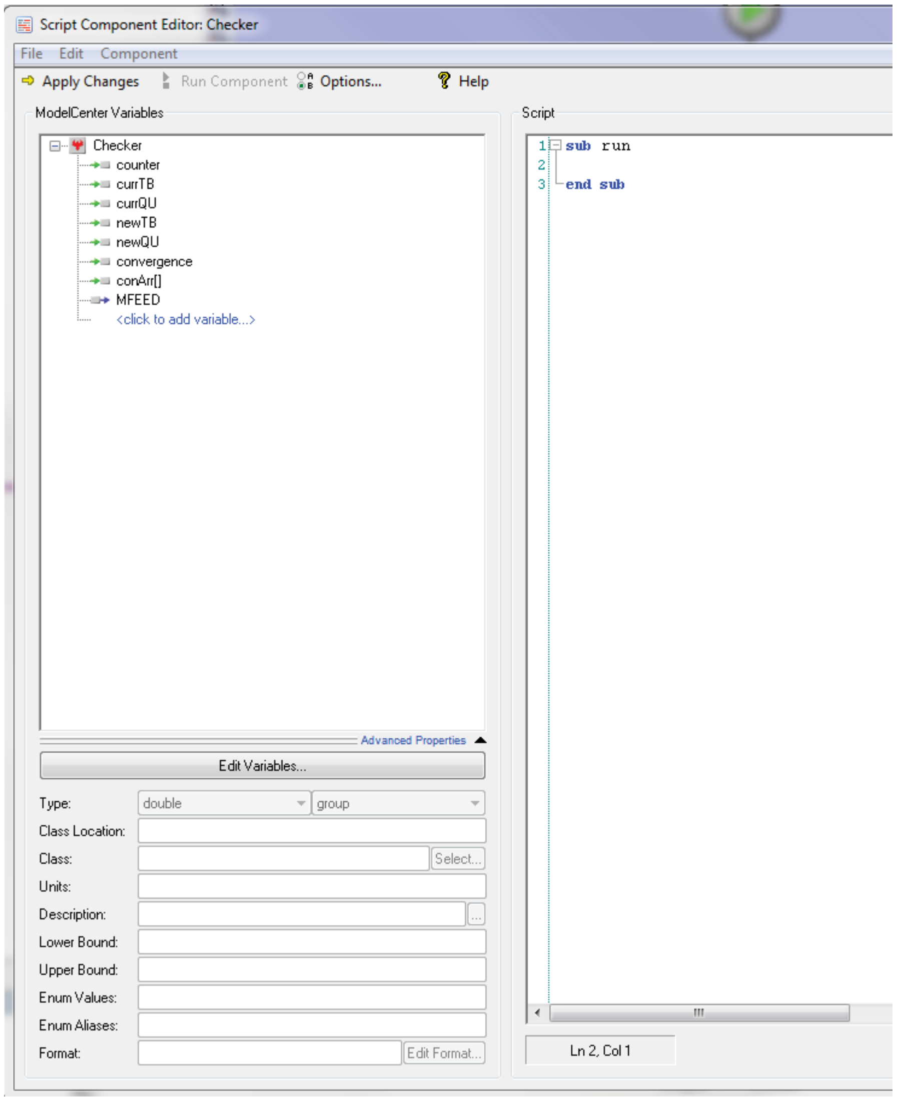


```
sub run

end sub
```

Language: VBScript   Script Timeout: 30

A pop-up window should come up in which you can write code and add variables. Note that the script language is defaulted to VBScript (the scripting version of Visual Basic), but you can easily select a different language from the drop-down menu. For this model, we are going to use VBScript.

First, set up the variables that are needed for this component. Variables that are needed of type double are newTB, newQU, currTB, currQU, counter, and MFEED. Also add the boolean variable convergence, and add a boolean array (denoted as boolean[]) called conArr. After setting up these variables your editor should look like this:



Checker
- counter
- currTB
- currQU
- newTB
- newQU
- convergence
- conArr[]
- MFEED
- <click to add variable...>

```
sub run

end sub
```

Type:	double / group
Class Location:	
Class:	
Units:	
Description:	
Lower Bound:	
Upper Bound:	
Enum Values:	
Enum Aliases:	
Format:	

Now we can start adding code to the script. The first block of code we are going to add in the component editor is shown below:

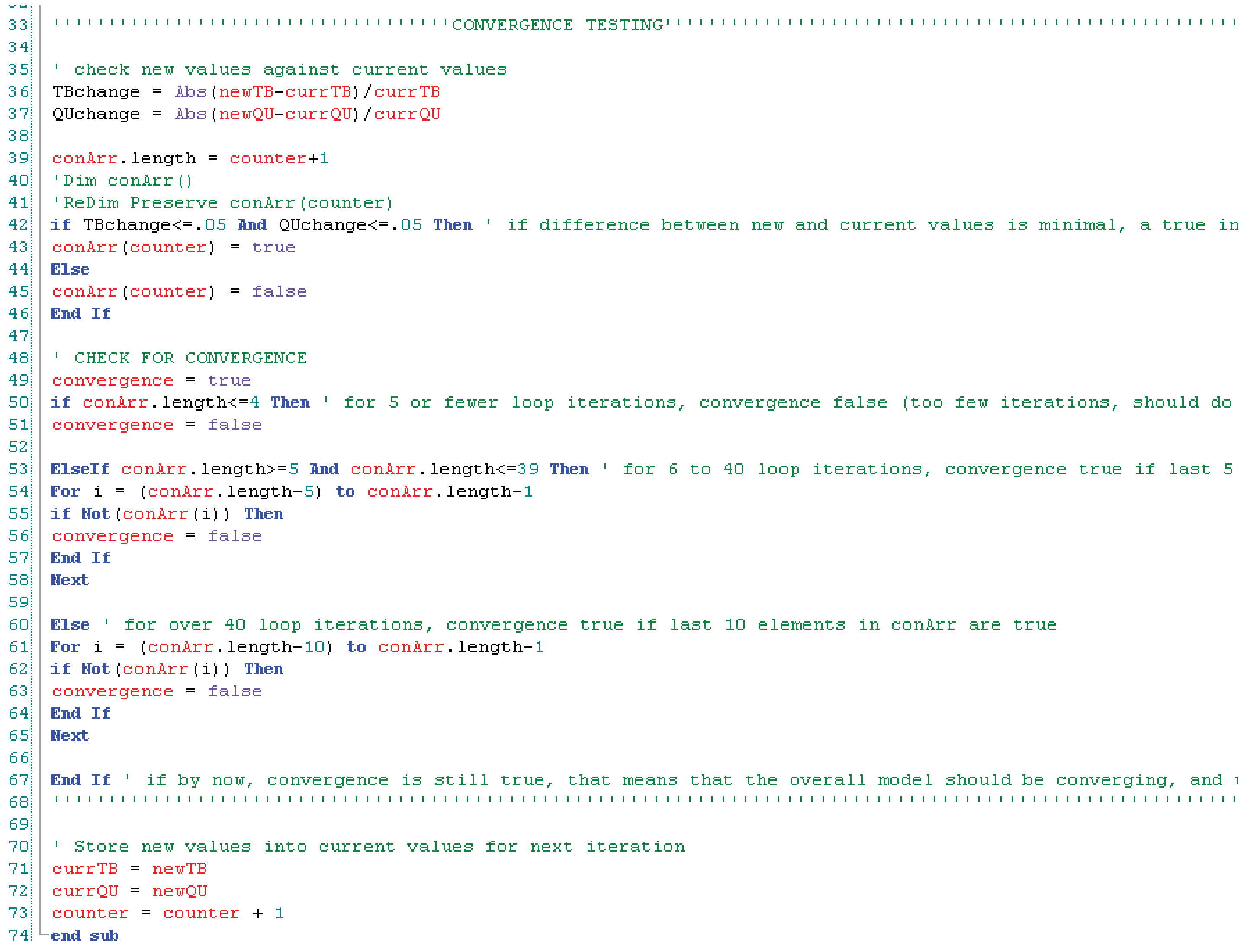
```
sub run
''''''''''''''''''Specify mass flux of slurry to cold cap, MFEED (kg/m^2s) '''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
if counter=0 Then
    MFEED = InputBox("Enter value for mass flux of slurry, MFEED (units of kg/m^2s)", "MFEED Specification")
    While Not(MFEED>0 And MFEED<0.1)
        MFEED = InputBox("Invalid value. MFEED must be a number between 0 and 0.1 kg/m^2s. Please enter a valid value.", "MFEED Specification")
    Wend
end if
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

This code prompts the user to specify a value for MFEED, the mass flux of slurry entering the cold cap, on the loop's first iteration (when counter is equal to 0). If the value is not between 0 and 0.1 kg/m^{2}s, the user is prompted to reenter a valid number. If the user wants to exit the model and no longer enter a value for MFEED, he/she will have to halt the model itself.

In the next block of code entered into the script, shown below, the values produced by STAR-CCM+ are checked to ensure that they are within reasonable bounds. If TB, the temperature at the bottom of the cold cap, is not between 873K and 1574K, a warning message is displayed to the user, and the value of TB is adjusted to one within bounds. Similarly, the value of QU, the heat transfer from the plenum space to the slurry, is checked to see if it is between 0 and 60 kW/m^2. If the value is negative, it is changed to a positive number, as the MATLAB model can only handle positive heat transfer values. If the value is positive but greater than 60 kW/m^2, it is adjusted to be 25.5 kW/m^2, which is reflective of a situation in which half the heat for evaporating and heating the slurry is supplied from the plenum space. Should the need arise, the lower and upper bounds used to perform bounds checking on these variables can be adjusted easily, and the same is true for the handling of out-of-bounds values.

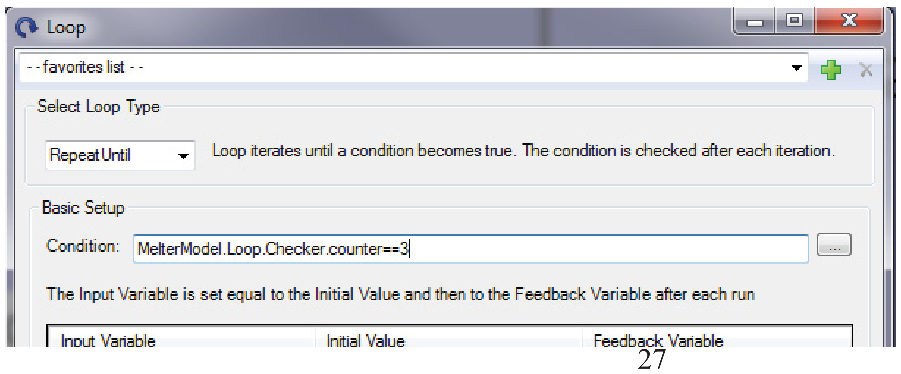
```
''''''''''''''''''''''''''''''''''''''''RANGE CHECKING'''''''''''''''''''''''''''''''''''''''''''''''''''
if Not(newTB>=873 And newTB<=1574) Then
MsgBox("Warning: TB value out of range. Now adjusting to a value within range.")
if newTB<873 Then
newTB = 873.15
Else
newTB = 1573.15
End If
End If

' make sure negative QU is turned into positive value for MATLAB
if newQU<0 Then
newQu = -1*newQu
End If

if Not(newQU<=60000) Then
MsgBox("Warning: QU value out of range. Now adjusting to a value within range.")
' then just assume that half the heat for evaporating and heating slurry is supplied from plenum space
newQU = 25500
End If
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
```

`The next block of code in the script component, shown below, controls how many times the loop runs.



```
''''''''''''''''''''''''''''''''''''''''CONVERGENCE TESTING'''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' check new values against current values
TBchange = Abs(newTB-currTB)/currTB
QUchange = Abs(newQU-currQU)/currQU

conArr.length = counter+1
'Dim conArr()
'ReDim Preserve conArr(counter)
if TBchange<=.05 And QUchange<=.05 Then ' if difference between new and current values is minimal, a true in
conArr(counter) = true
Else
conArr(counter) = false
End If

' CHECK FOR CONVERGENCE
convergence = true
if conArr.length<=4 Then ' for 5 or fewer loop iterations, convergence false (too few iterations, should do
convergence = false

ElseIf conArr.length>=5 And conArr.length<=39 Then ' for 6 to 40 loop iterations, convergence true if last 5
For i = (conArr.length-5) to conArr.length-1
if Not(conArr(i)) Then
convergence = false
End If
Next

Else ' for over 40 loop iterations, convergence true if last 10 elements in conArr are true
For i = (conArr.length-10) to conArr.length-1
if Not(conArr(i)) Then
convergence = false
End If
Next

End If ' if by now, convergence is still true, that means that the overall model should be converging, and
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''

' Store new values into current values for next iteration
currTB = newTB
currQU = newQU
counter = counter + 1
end sub
```

There are two ways to control the number of iterations performed by the loop: either the number of iterations can be specified explicitly, or the loop can be made to run as many times as it takes for the model to converge upon a solution. For the first method, a variable called "counter" is updated every time the loop runs. In the Loop component's editor, the "Repeat Until" condition can be altered to cap the value of "counter" at a certain point. For instance, if we want the loop to only run for three iterations, we would make the loop's "Condition" look like the following:



Repeat Until

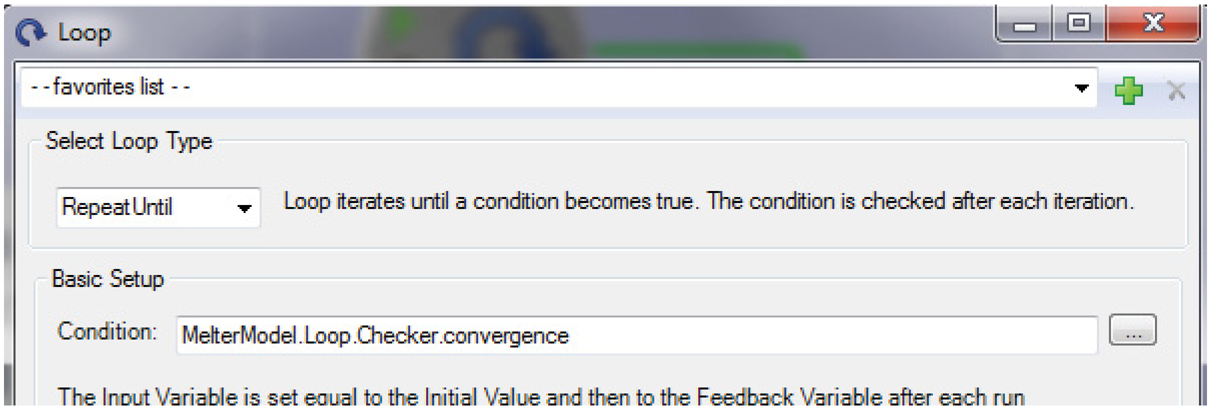
Loop iterates until a condition becomes true. The condition is checked after each iteration.

Condition: MelterModel.Loop.Checker.counter==3

The Input Variable is set equal to the Initial Value and then to the Feedback Variable after each run

Input Variable	Initial Value	Feedback Variable

This set-up will be different if you want to run the model until it converges. In this case, the model will run until the variable "convergence" is true, as shown below.



Repeat Until — Loop iterates until a condition becomes true. The condition is checked after each iteration.

Condition: MelterModel.Loop.Checker.convergence

The Input Variable is set equal to the Initial Value and then to the Feedback Variable after each run

The script component has a very precise method of checking for convergence. The script component stores old values for TB and QU (currTB and currQU) that are from the previous iteration. The component also stores the new value for each variable (newTB and newQU) that has just been calculated by STAR-CCM+ in the current iteration. The change between the old and new values is then calculated. If the change is less than 5% for both TB and QU, a boolean "true" is added to the boolean array conArr. If the model has been running for anywhere between 6 and 40 iterations (i.e., the length of conArr is either 5 or 39), and the last five elements of conArr are all "true", then the convergence variable is changed to "true," and the loop stops running. Similarly, if the loop has been running for over 40 iterations, the convergence variable is changed to "true" if the last 10 values in conArr are all "true." Finally, at the end of the script, the new values of QU and TB are stored in currTB and currQU so that they become the old values in preparation for the next iteration. Note that the variable counter is also incremented by one.

2.5 Adding a Script Component that Uploads Pictures to MICE Website

We can upload the image files outputted by STAR-CCM+ to the MICE website (http://inlteam2010:3666/sites/GEMS/MICE). This makes use of a Windows PowerShell script (copyFileToSharepoint.ps1, developed by Brant Peery at INL) that can upload images to a website.

First, go to the Start menu, and find Windows PowerShell (x86). Right-click on it, and select "Run as administrator." This should bring up a blue PowerShell terminal. In this terminal, first type in the following, and then press enter:

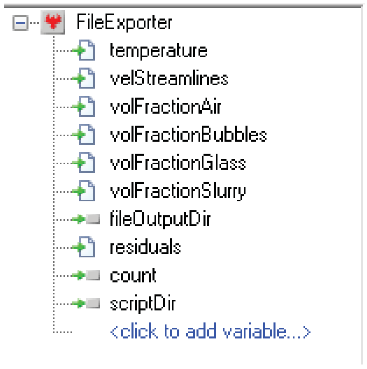
```
get-executionpolicy
```

If, after pressing enter, the terminal displays "remotesigned," you do not need to do anything else because this means that you will be able to run PowerShell scripts on your computer. If, however, the terminal displays "restricted," you will not be able to run PowerShell scripts. In this case, you need to manually set the execution policy to "remotesigned." To do this, type the following into the terminal, and then press enter:

```
set-executionpolicy remotesigned
```

After you press enter, it should ask you to confirm whether or not you would like to change the execution policy. Type in "y" for yes, and press enter. Now your execution policy should be what we need.

Now we will actually create the script component in ModelCenter that will execute the PowerShell script. Drag and drop a Script component into the model, just after the melter component in the loop. This process should be similar to how we created the script component in the previous section. VBScript will be used once again as the language for this script. The variables that need to be added to this script are shown below. The variables temperature, velStreamlines, volFractionAir, volFractionBubbles, volFractionGlass, volFractionSlurry, and residuals are all file variables. The variables fileOutputDir and scriptDir are both strings. The variable count is of type int.


```
FileExporter
    temperature
    velStreamlines
    volFractionAir
    volFractionBubbles
    volFractionGlass
    volFractionSlurry
    fileOutputDir
    residuals
    count
    scriptDir
    <click to add variable...>
```

The code for this script is shown below:

```
Option Explicit

dim wsh
Set wsh = CreateObject("Wscript.shell")

sub run
dim filename
  filename = fileOutputDir & "temperature.png"
  temperature.isBinary = true
  temperature.toFile(filename)
  uploadfile filename, "temperature" & count & ".png"

  filename = fileOutputDir & "velStreamlines.png"
    velStreamlines.isBinary = true
  velStreamlines.toFile(filename)
  uploadfile filename, "velStreamlines" & count & ".png"

  filename = fileOutputDir & "volFractionAir.png"
    volFractionAir.isBinary = true
  volFractionAir.toFile(filename)
  uploadfile filename, "volFractionAir" & count & ".png"

  filename = fileOutputDir & "volFractionBubbles.png"
    volFractionBubbles.isBinary = true
  volFractionBubbles.toFile(filename)
  uploadfile filename, "volFractionBubbles" & count & ".png"

  filename = fileOutputDir & "volFractionGlass.png"
    volFractionGlass.isBinary = true
  volFractionGlass.toFile(filename)
  uploadfile filename, "volFractionGlass" & count & ".png"

  filename = fileOutputDir & "volFractionSlurry.png"
    volFractionSlurry.isBinary = true
  volFractionSlurry.toFile(filename)
  uploadfile filename, "volFractionSlurry" & count & ".png"

  filename = fileOutputDir & "residuals.png"
    residuals.isBinary = true
  residuals.toFile(filename)
  uploadfile filename, "residuals" & count & ".png"

end sub

sub uploadFile(filePath, rename)
    dim sharepointURL, scriptPath, cmd
    sharepointURL = "http://inlteam2010:3666/sites/GEMS/MICE/MICE%20Output"
    scriptPath = scriptDir & "copyFileToSharepoint.ps1"

    cmd = "powershell -NonInteractive -file " & scriptPath & " -FilePath " & filePath & " -SharepointURL " & sharepointURL
    cmd = cmd & " -rename " & rename

    'msgbox cmd
    wsh.run cmd, 0
end sub
```

In this script, an object is first created that represents the shell/terminal in which the PowerShell script will be executed. Next, all the files are uploaded one-by-one onto the MICE website. Notice that for each file, the absolute path to the file is specified by concatenating the fileOutputDir variable and the filename together. The file variable is then set to binary and saved to the absolute pathname that was just created.

Next, the "uploadfile" method is called that executes the PowerShell script by running the string stored in the "cmd" variable in the "wsh" shell object. Notice that the variable "count" is used to append the iteration number onto the filename; this allows pictures to be distinguishable from one another on the MICE website. The count variable in this script component will be linked with the counter variable in the Checker script component.

It is also important to note that you will need to set up the values for the fileOutputDir and scriptDir variables before you can run this component successfully. The scriptDir variable must be the full path to the folder that contains the "copyFileToSharepoint.ps1" script. For instance, I have this script saved to my Desktop, so I changed the value of scriptDir to be "c:\Users\agarv2\Desktop\". I have set the fileOutputDir variable to be my temp directory (c:\temp\). You do not have to change this as long as you have a temp directory on your C drive. This directory is where pictures will be temporarily saved before they are uploaded onto the website.

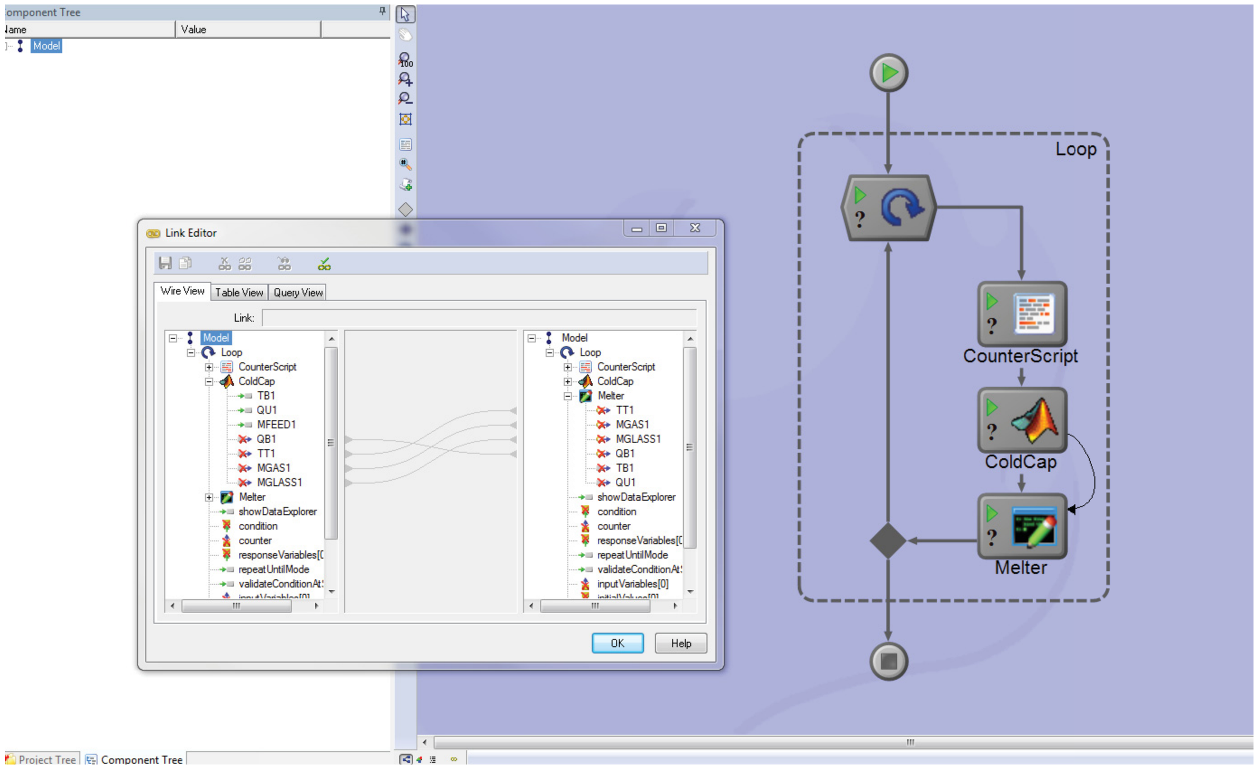
Whenever this component runs, you should be able to go to the MICE website, navigate to the MICE Output page, and see the pictures.

2.6 Linking Components Together and Setting Loop Properties

Note that you can rename components to better represent what each component is.

Now we are going to link the variables in the components together. If you hover your mouse around the center of the Matlab component, a link symbol should appear. Click and drag that link to the STAR-CCM+ melter component. This should automatically link variables of the same name together. In this case, it links the outputs of MATLAB to the inputs of STAR-CCM+ since we set them up as having the same name. It should also pull up a link editor so you can see the links more clearly.

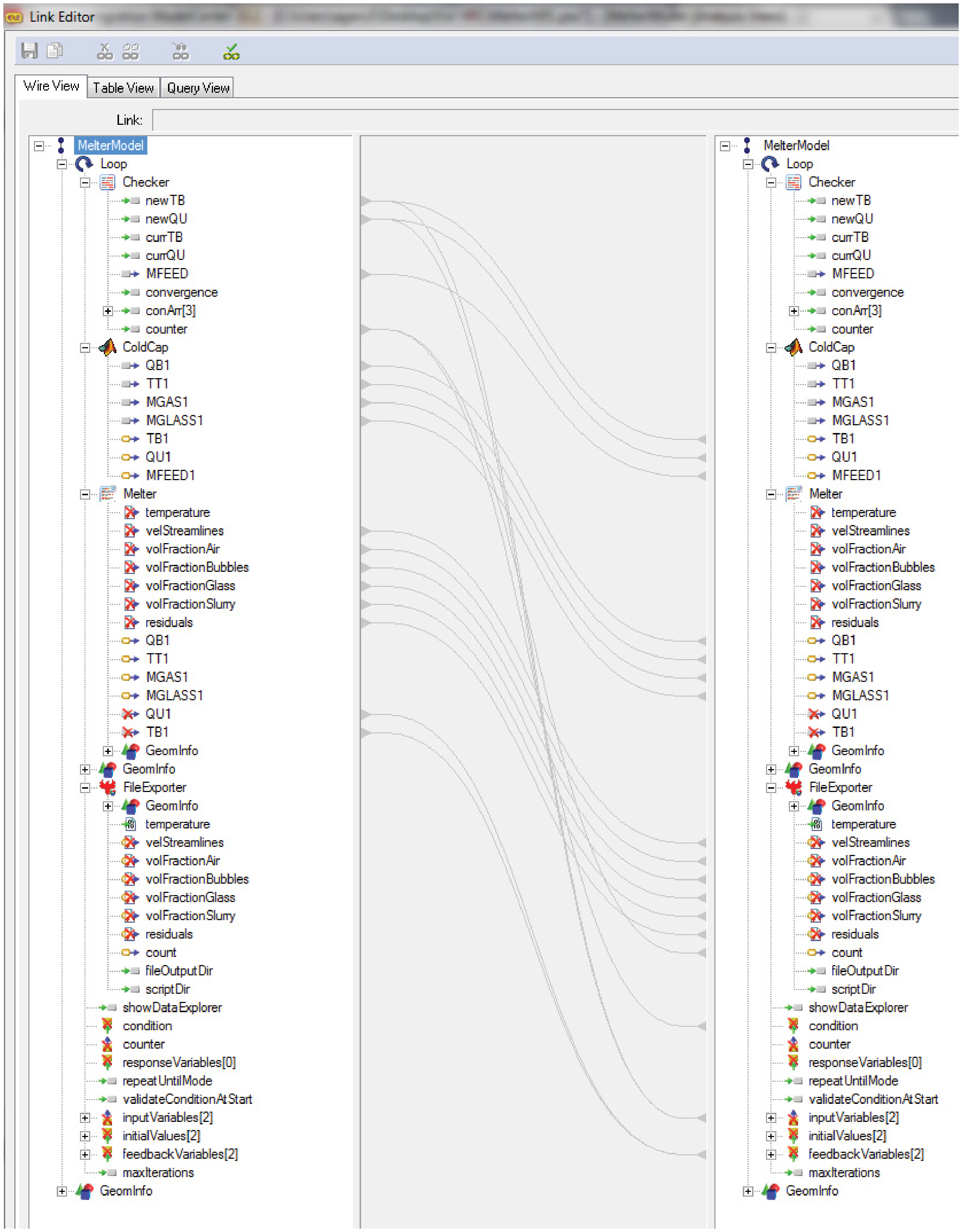
Once done, you should now have a black arrow going from the MATLAB component to the STAR component, and your link editor should look the way it does in the picture below.



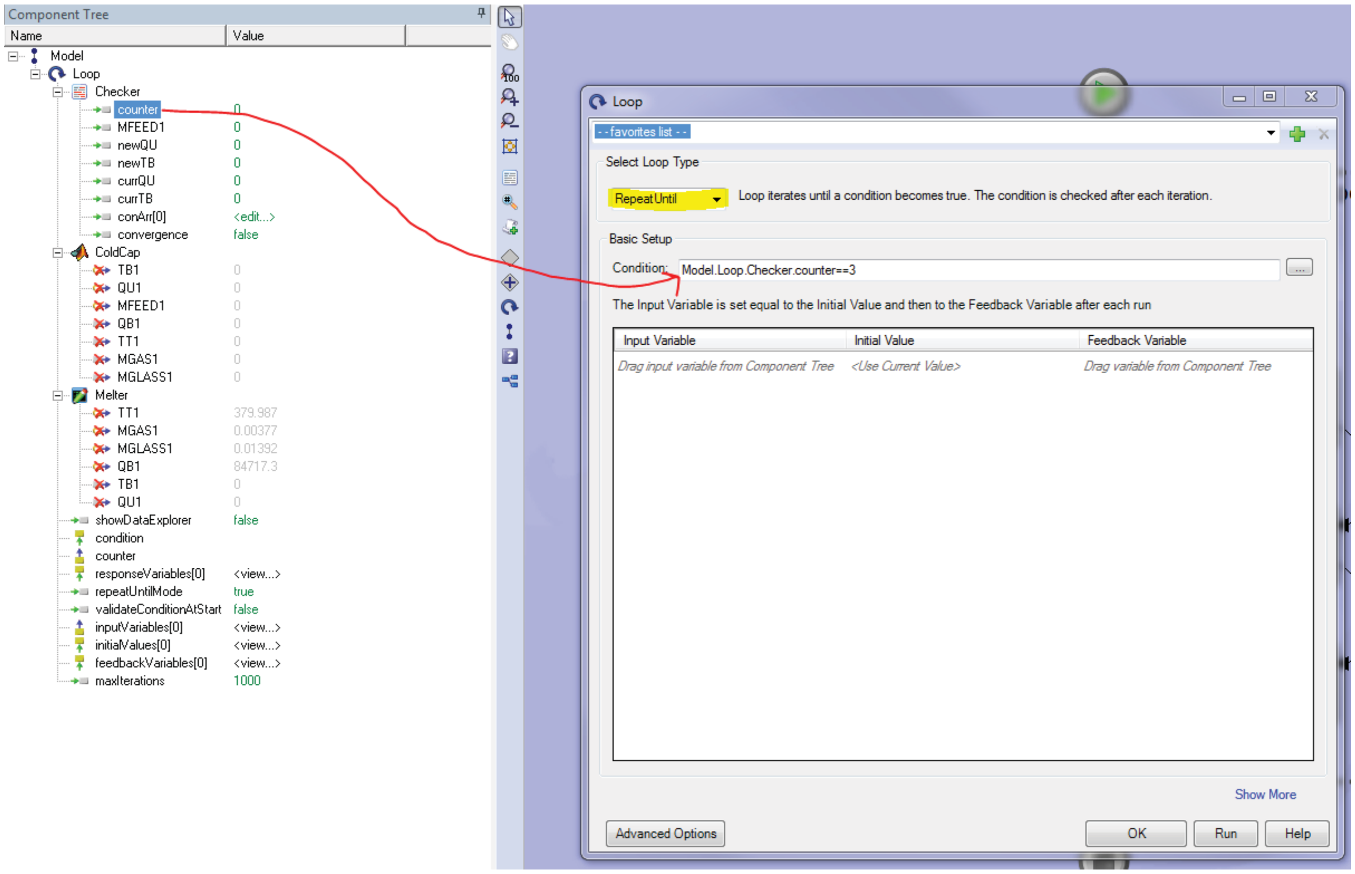
We are also going to link the script component and the MATLAB component together. We want the MFEED, newQU and newTB values of the script component to flow to the MFEED1, QU1 and TB1 values of the MATLAB component. Because these variables do not have the same names, we cannot create the links using the method used before to link the MATLAB and STAR-CCM+ components. Instead, we will have to open the link editor by clicking on the "∞" symbol in the toolbar in the bottom left of the model window. To connect MFEED in the script component to MFEED1 in the MATLAB component, click on MFEED in the left pane and drag it to MFEED1 in the right pane. A link should then be seen between the two variables. Do this for the rest of the variables that are to be linked between the script component and MATLAB.

Now, using the link editor, link the counter variable in the Checker component with the count variable in the FileExporter component. The last few links you then need to create are between the Melter component and the FileExporter component: all the file variables need to be linked to each other. Therefore, the temperature file variable in the Melter component needs to be linked to the temperature file variable in the FileExporter component, and so on.

Once you have linked all the variables, your link editor should look like this:



Now we will finally set up the loop properties. Double-click on the Loop component to bring up its editor. Remember that you can control how many times the loop runs in different ways; in this case, let's explicitly specify that the loop should run 3 times. For the condition, drag the counter variable from the Component Tree to the condition box, and specify that the loop should repeat until the counter variable equals 3. To do this, simply type in "==3" after the counter variable name that appeared in the box.


Component Tree

Name	Value
Model	
Loop	
Checker	
counter	0
MFEED1	0
newQU	0
newTB	0
currQU	0
currTB	0
conArr[0]	<edit...>
convergence	false
ColdCap	
TB1	0
QU1	0
MFEED1	0
QB1	0
TT1	0
MGAS1	0
MGLASS1	0
Melter	
TT1	379.987
MGAS1	0.00377
MGLASS1	0.01392
QB1	84717.3
TB1	0
QU1	0
showDataExplorer	false
condition	
counter	
responseVariables[0]	<view...>
repeatUntilMode	true
validateConditionAtStart	false
inputVariables[0]	<view...>
initialValues[0]	<view...>
feedbackVariables[0]	<view...>
maxIterations	1000

Loop

-- favorites list --

Select Loop Type

Repeat Until — Loop iterates until a condition becomes true. The condition is checked after each iteration.

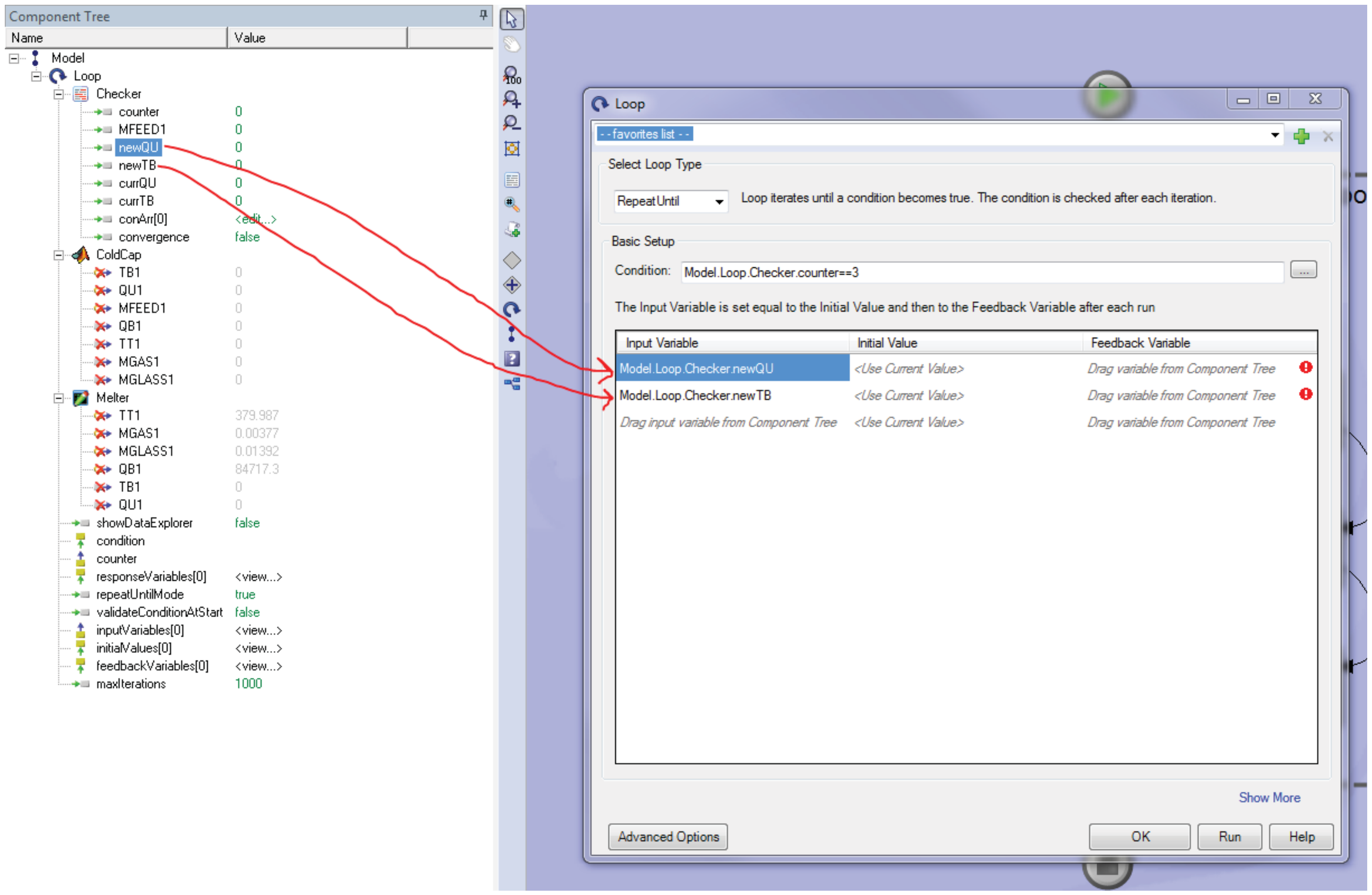
Basic Setup

Condition: Model.Loop.Checker.counter==3

The Input Variable is set equal to the Initial Value and then to the Feedback Variable after each run

Input Variable	Initial Value	Feedback Variable
Drag input variable from Component Tree	<Use Current Value>	Drag variable from Component Tree

We now need the outputs of STAR-CCM+ to become inputs for the script component upon every loop iteration. The script component will perform bounds checking of these values and then pass them along to MATLAB. Therefore, we are going to drag the variables newQU and newTB in the script component over to the “Input Variable” section.


Component Tree

Name	Value
Model	
Loop	
Checker	
counter	0
MFEED1	0
newQU	0
newTB	0
currQU	0
currTB	0
conArr[0]	<edit...>
convergence	false
ColdCap	
TB1	0
QU1	0
MFEED1	0
QB1	0
TT1	0
MGAS1	0
MGLASS1	0
Melter	
TT1	379.987
MGAS1	0.00377
MGLASS1	0.01392
QB1	84717.3
TB1	0
QU1	0
showDataExplorer	false
condition	
counter	
responseVariables[0]	<view...>
repeatUntilMode	true
validateConditionAtStart	false
inputVariables[0]	<view...>
initialValues[0]	<view...>
feedbackVariables[0]	<view...>
maxIterations	1000

Loop

- - favorites list - -

Select Loop Type

Repeat Until — Loop iterates until a condition becomes true. The condition is checked after each iteration.

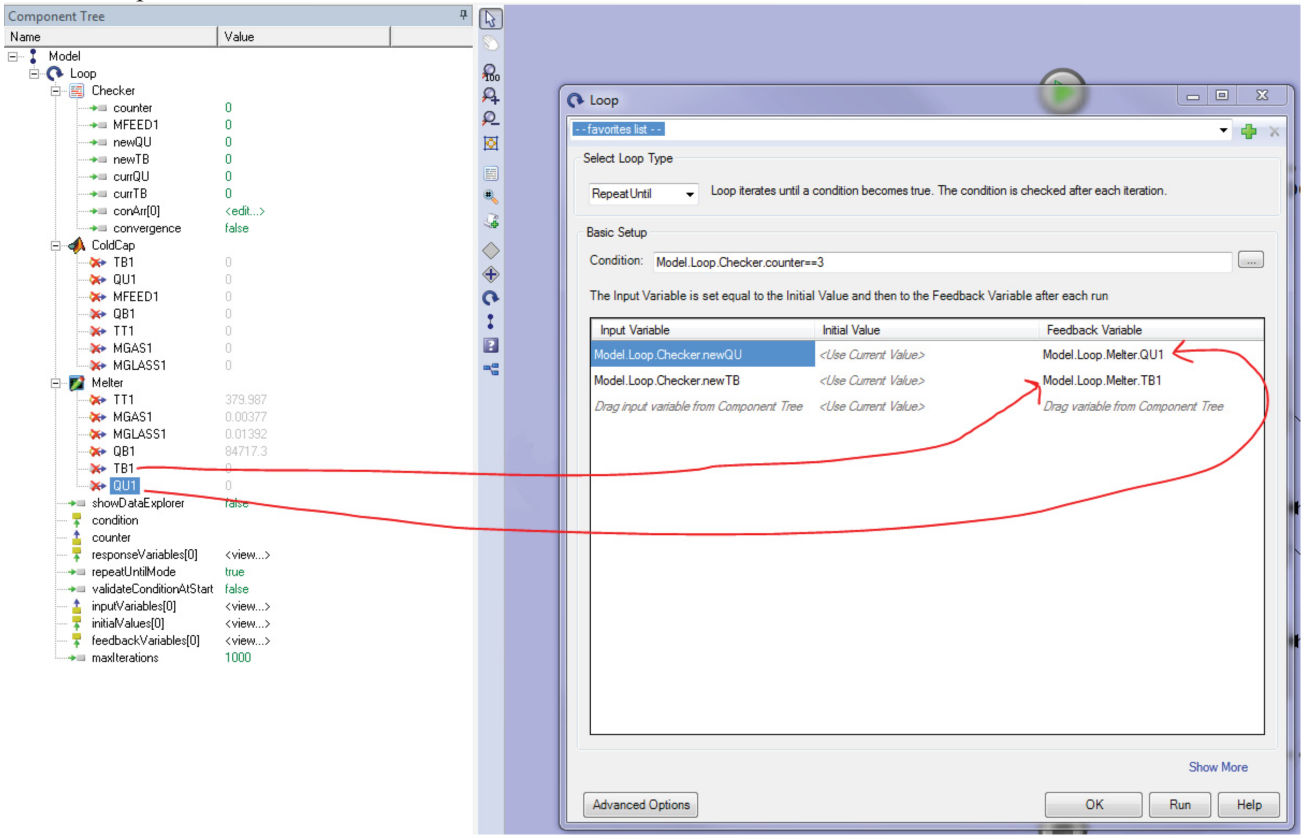
Basic Setup

Condition: Model.Loop.Checker.counter==3

The Input Variable is set equal to the Initial Value and then to the Feedback Variable after each run

Input Variable	Initial Value	Feedback Variable
Model.Loop.Checker.newQU	<Use Current Value>	Drag variable from Component Tree
Model.Loop.Checker.newTB	<Use Current Value>	Drag variable from Component Tree
Drag input variable from Component Tree	<Use Current Value>	Drag variable from Component Tree

Now we will set the "Feedback Variable" for these two input variables to be the outputs coming from the STAR component.


Component Tree

Name	Value
Model	
Loop	
Checker	
counter	0
MFEED1	0
newQU	0
newTB	0
currQU	0
currTB	0
conArr[0]	<edit...>
convergence	false
ColdCap	
TB1	0
QU1	0
MFEED1	0
QB1	0
TT1	0
MGAS1	0
MGLASS1	0
Melter	
TT1	379.987
MGAS1	0.00377
MGLASS1	0.01392
QB1	84717.3
TB1	0
QU1	0
showDataExplorer	false
condition	
counter	
responseVariables[0]	<view...>
repeatUntilMode	true
validateConditionAtStart	false
inputVariables[0]	<view...>
initialValues[0]	<view...>
feedbackVariables[0]	<view...>
maxIterations	1000

Loop

- - favorites list - -

Select Loop Type

RepeatUntil — Loop iterates until a condition becomes true. The condition is checked after each iteration.

Basic Setup

Condition: Model.Loop.Checker.counter==3

The Input Variable is set equal to the Initial Value and then to the Feedback Variable after each run

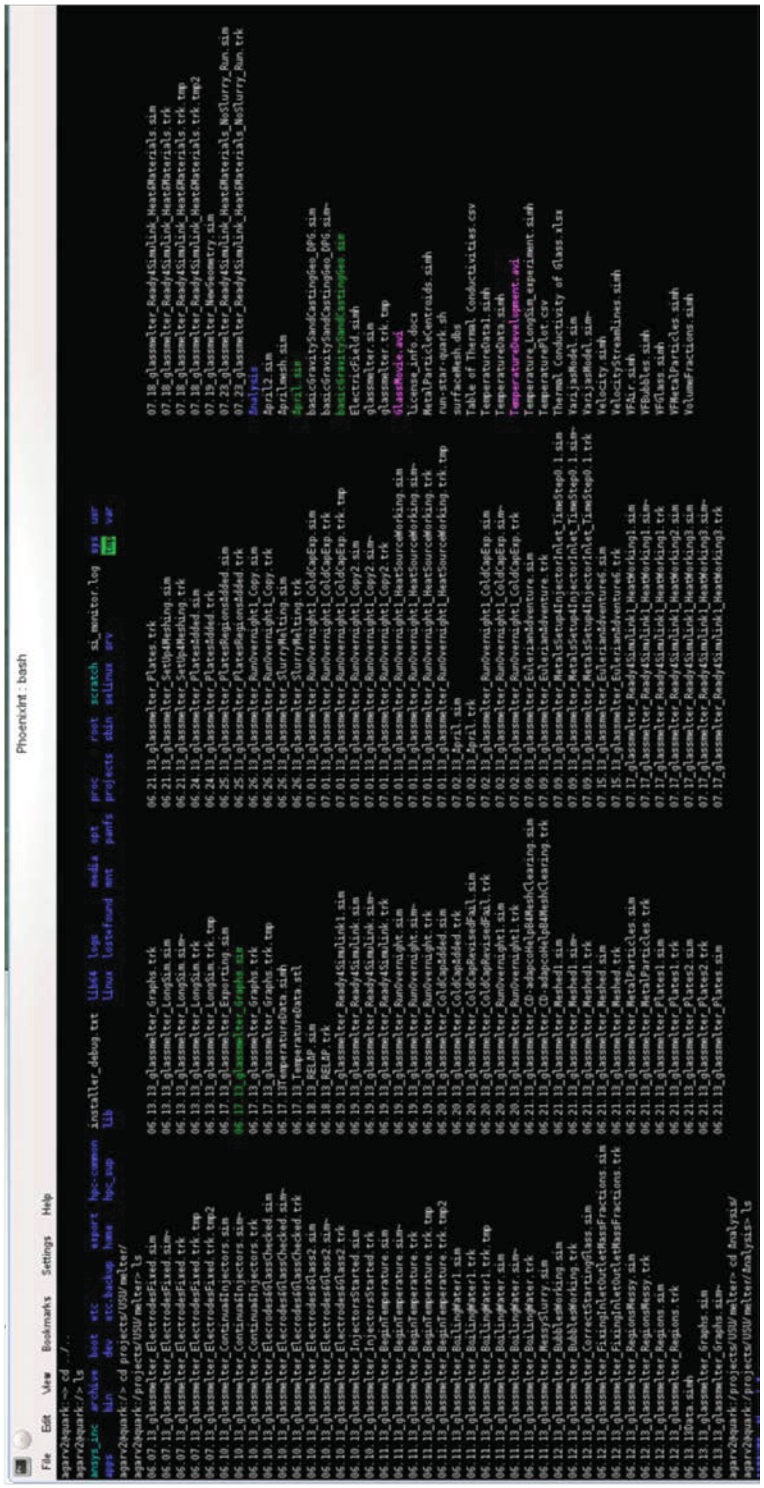
Input Variable	Initial Value	Feedback Variable
Model.Loop.Checker.newQU	<Use Current Value>	Model.Loop.Melter.QU1
Model.Loop.Checker.newTB	<Use Current Value>	Model.Loop.Melter.TB1
Drag input variable from Component Tree	<Use Current Value>	Drag variable from Component Tree

Save your changes.

We are now done setting up the model. It should run as is.

2.7 Setting up the Model to Run STAR-CCM+ on HPC

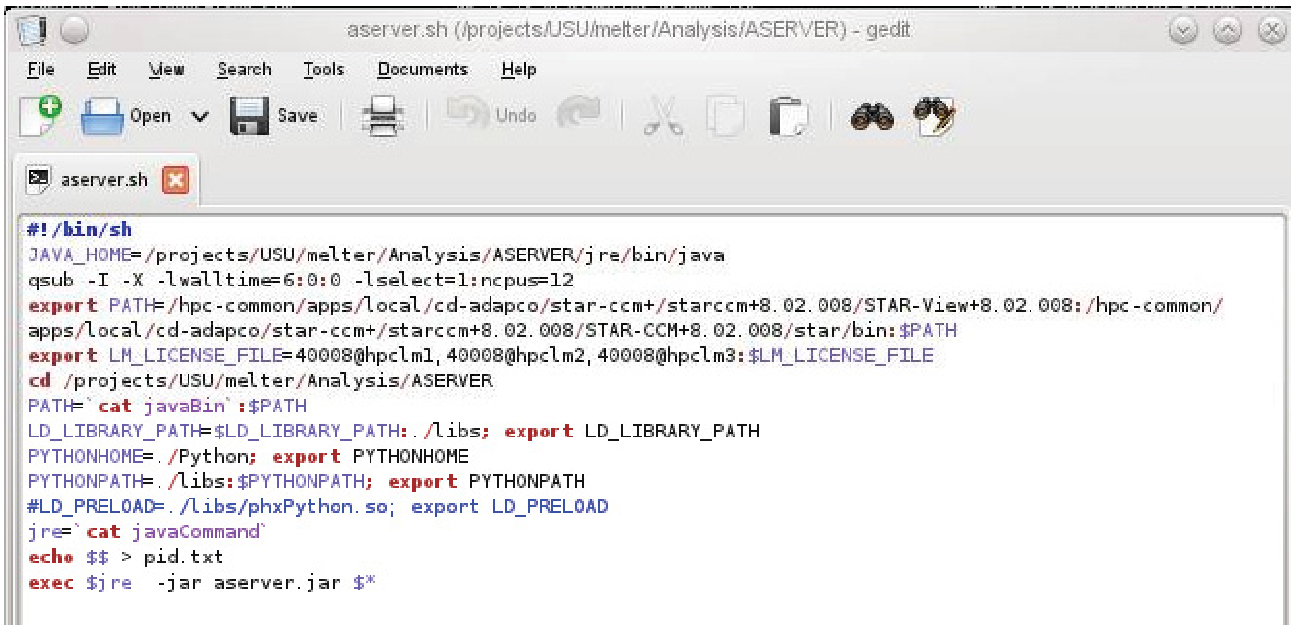
In order to be able to run STAR-CCM+ remotely on HPC instead of on your desktop, you will need to have AnalysisServer installed on your HPC. Right now, this is set up in the /projects/USU/melter folder under the Analysis directory. (The picture below might be slightly unclear, but you should be able to navigate to the Analysis directory easily on HPC, since the projects/USU/melter folder is accessible to all of us).



```
agarv2@quark:~> cd ../..
agarv2@quark:/> ls
ansys_inc  archive  boot  etc         export  hpc-common  installer_debug.txt  lib64  logs        media  opt    proc      root  scratch  si_monitor.log  sys  usr
apps       bin      dev   etc.backup  home    hpc_sup     lib                  linux  lost+found  mnt    panfs  projects  sbin  selinux  srv             tmp  var
agarv2@quark:/> cd projects/USU/melter/
agarv2@quark:/projects/USU/melter> ls
06.07.13_glassmelter_ElectrodesFixed.sim                06.13.13_glassmelter_Graphs.trk                      06.21.13_glassmelter_Plates.trk                                 07.18_glassmelter_Ready4Simulink_Heat&Materials.sim
06.07.13_glassmelter_ElectrodesFixed.sim~               06.13.13_glassmelter_LongSim.sim                     06.21.13_glassmelter_SetUp4Meshing.sim                          07.18_glassmelter_Ready4Simulink_Heat&Materials.trk
06.07.13_glassmelter_ElectrodesFixed.trk                06.13.13_glassmelter_LongSim.sim~                    06.21.13_glassmelter_SetUp4Meshing.trk                          07.18_glassmelter_Ready4Simulink_Heat&Materials.trk.tmp
06.07.13_glassmelter_ElectrodesFixed.trk.tmp            06.13.13_glassmelter_LongSim.trk                     06.24.13_glassmelter_PlatesAdded.sim                            07.18_glassmelter_Ready4Simulink_Heat&Materials.trk.tmp2
06.07.13_glassmelter_ElectrodesFixed.trk.tmp2           06.13.13_glassmelter_LongSim.trk.tmp                 06.24.13_glassmelter_PlatesAdded.trk                            07.19_glassmelter_NewGeometry.sim
06.10.13_glassmelter_Continua&Injectors.sim             06.17.13_glassmelter_Exporting.sim                   06.25.13_glassmelter_PlatesRegionsAdded.sim                     07.23_glassmelter_Ready4Simulink_Heat&Materials_NoSlurry_Run.sim
06.10.13_glassmelter_Continua&Injectors.sim~            06.17.13_glassmelter_Graphs.sim                      06.25.13_glassmelter_PlatesRegionsAdded.trk                     07.23_glassmelter_Ready4Simulink_Heat&Materials_NoSlurry_Run.trk
06.10.13_glassmelter_Continua&Injectors.trk             06.17.13_glassmelter_Graphs.trk                      06.26.13_glassmelter_RunOvernight1_Copy.sim                     Analysis
06.10.13_glassmelter_Elecrodes&GlassChecked.sim         06.17.13_glassmelter_Graphs.trk.tmp                  06.26.13_glassmelter_RunOvernight1_Copy.trk                     April2.sim
06.10.13_glassmelter_Elecrodes&GlassChecked.sim~        06.17.13TemperatureData.simh                         06.26.13_glassmelter_SlurryMelting.sim                          Aprilmesh.sim
06.10.13_glassmelter_Elecrodes&GlassChecked.trk         06.17.13_TemperatureData.stl                         06.26.13_glassmelter_SlurryMelting.trk                          April.sim
06.10.13_glassmelter_Electrodes&Glass2.sim              06.18.13_RELAP.sim                                   07.01.13_glassmelter_RunOvernight1_ColdCapExp.sim               basicGravitySandCastingGeo_DPG.sim
06.10.13_glassmelter_Electrodes&Glass2.sim~             06.18.13_RELAP.trk                                   07.01.13_glassmelter_RunOvernight1_ColdCapExp.trk               basicGravitySandCastingGeo_DPG.sim~
06.10.13_glassmelter_Electrodes&Glass2.trk              06.19.13_glassmelter_Ready4Simulink1.sim             07.01.13_glassmelter_RunOvernight1_ColdCapExp.trk.tmp           basicGravitySandCastingGeo.sim
06.10.13_glassmelter_InjectorsStarted.sim               06.19.13_glassmelter_Ready4Simulink.sim              07.01.13_glassmelter_RunOvernight1_Copy2.sim                    ElectricField.simh
06.10.13_glassmelter_InjectorsStarted.trk               06.19.13_glassmelter_Ready4Simulink.sim~             07.01.13_glassmelter_RunOvernight1_Copy2.sim~                   glassmelter.sim
06.11.13_glassmelter_BeginTemperature.sim               06.19.13_glassmelter_Ready4Simulink.trk              07.01.13_glassmelter_RunOvernight1_Copy2.trk                    glassmelter.trk.tmp
06.11.13_glassmelter_BeginTemperature.sim~              06.19.13_glassmelter_RunOvernight.sim                07.01.13_glassmelter_RunOvernight1_HeatSourceWorking.sim        GlassMovie.avi
06.11.13_glassmelter_BeginTemperature.trk               06.19.13_glassmelter_RunOvernight.sim~               07.01.13_glassmelter_RunOvernight1_HeatSourceWorking.sim~       license_info.docx
06.11.13_glassmelter_BeginTemperature.trk.tmp           06.19.13_glassmelter_RunOvernight.trk                07.01.13_glassmelter_RunOvernight1_HeatSourceWorking.trk        MetalParticleCentroids.simh
06.11.13_glassmelter_BeginTemperature.trk.tmp2          06.20.13_glassmelter_ColdCapAdded.sim                07.01.13_glassmelter_RunOvernight1_HeatSourceWorking.trk.tmp    run-star-quark.sh
06.11.13_glassmelter_BoilingWater1.sim                  06.20.13_glassmelter_ColdCapAdded.trk                07.02.13_April.sim                                              surfaceMesh.dbs
06.11.13_glassmelter_BoilingWater1.trk                  06.20.13_glassmelter_ColdCapRevisedFail.sim          07.02.13_April.trk                                              Table of Thermal Conductivities.csv
06.11.13_glassmelter_BoilingWater1.trk.tmp              06.20.13_glassmelter_ColdCapRevisedFail.trk          07.02.13_glassmelter_RunOvernight1_ColdCapExp.sim               TemperatureData1.simh
06.11.13_glassmelter_BoilingWater.sim                   06.20.13_glassmelter_RunOvernight1.sim               07.02.13_glassmelter_RunOvernight1_ColdCapExp.sim~              TemperatureData.simh
06.11.13_glassmelter_BoilingWater.sim~                  06.20.13_glassmelter_RunOvernight1.trk               07.02.13_glassmelter_RunOvernight1_ColdCapExp.trk               TemperatureDevelopment.avi
06.11.13_glassmelter_BoilingWater.trk                   06.21.13_glassmelter_CD-adapcoHelpB4MeshClearing.sim 07.09.13_glassmelter_EulerianAdventure.sim                      Temperature_LongSim_experiment.simh
06.11.13_glassmelter_MessySlurry.sim                    06.21.13_glassmelter_CD-adapcoHelpB4MeshClearing.trk 07.09.13_glassmelter_EulerianAdventure.trk                      TemperaturePlot.csv
06.12.13_glassmelter_BubblesWorking.sim                 06.21.13_glassmelter_Meshed1.sim                     07.09.13_glassmelter_MetalsSetup4InjectorInlet_TimeStep0.1.sim  Thermal Conductivity of Glass.xlsx
06.12.13_glassmelter_BubblesWorking.trk                 06.21.13_glassmelter_Meshed1.sim~                    07.09.13_glassmelter_MetalsSetup4InjectorInlet_TimeStep0.1.sim~ VarijasModel.sim
06.12.13_glassmelter_CorrectStartingGlass.sim           06.21.13_glassmelter_Meshed1.trk                     07.09.13_glassmelter_MetalsSetup4InjectorInlet_TimeStep0.1.trk  VarijasModel.sim~
06.12.13_glassmelter_FixingInletOutletMassFractions.sim 06.21.13_glassmelter_Meshed.sim                      07.15.13_glassmelter_EulerianAdventure6.sim                     Velocity.simh
06.12.13_glassmelter_FixingInletOutletMassFractions.trk 06.21.13_glassmelter_Meshed.trk                      07.15.13_glassmelter_EulerianAdventure6.trk                     VelocityStreamlines.simh
06.12.13_glassmelter_RegionsMessy.sim                   06.21.13_glassmelter_MetalParticles.sim              07.17_glassmelter_Ready4Simulink1_HeatWorking1.sim              VFAir.simh
06.12.13_glassmelter_RegionsMessy.trk                   06.21.13_glassmelter_MetalParticles.trk              07.17_glassmelter_Ready4Simulink1_HeatWorking1.sim~             VFBubbles.simh
06.12.13_glassmelter_Regions.sim                        06.21.13_glassmelter_Plates1.sim                     07.17_glassmelter_Ready4Simulink1_HeatWorking1.trk              VFGlass.simh
06.12.13_glassmelter_Regions.trk                        06.21.13_glassmelter_Plates1.trk                     07.17_glassmelter_Ready4Simulink1_HeatWorking2.sim              VFMetalParticles.simh
06.13.13Data.simh                                       06.21.13_glassmelter_Plates2.sim                     07.17_glassmelter_Ready4Simulink1_HeatWorking3.sim              VolumeFractions.simh
06.13.13_glassmelter_Graphs.sim                         06.21.13_glassmelter_Plates2.trk                     07.17_glassmelter_Ready4Simulink1_HeatWorking3.sim~
06.13.13_glassmelter_Graphs.sim~                        06.21.13_glassmelter_Plates.sim                      07.17_glassmelter_Ready4Simulink1_HeatWorking3.trk
agarv2@quark:/projects/USU/melter> cd Analysis/
agarv2@quark:/projects/USU/melter/Analysis> ls
```

Everything should already be set up, but an important file to know about is “aserver.sh.” This file can be used to set up the environment in which STAR-CCM+ will run. More specifically, this is where you want to load the module/version of STAR-CCM+ that you would like the simulation to run in. This file is located in the /projects/USU/melter/Analysis/ASERVER directory.

The contents of “aserver.sh” are shown below:



```
#!/bin/sh
JAVA_HOME=/projects/USU/melter/Analysis/ASERVER/jre/bin/java
qsub -I -X -lwalltime=6:0:0 -lselect=1:ncpus=12
export PATH=/hpc-common/apps/local/cd-adapco/star-ccm+/starccm+8.02.008/STAR-View+8.02.008:/hpc-common/
apps/local/cd-adapco/star-ccm+/starccm+8.02.008/STAR-CCM+8.02.008/star/bin:$PATH
export LM_LICENSE_FILE=40008@hpclm1,40008@hpclm2,40008@hpclm3:$LM_LICENSE_FILE
cd /projects/USU/melter/Analysis/ASERVER
PATH=`cat javaBin`:$PATH
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./libs; export LD_LIBRARY_PATH
PYTHONHOME=./Python; export PYTHONHOME
PYTHONPATH=./libs:$PYTHONPATH; export PYTHONPATH
#LD_PRELOAD=./libs/phxPython.so; export LD_PRELOAD
jre=`cat javaCommand`
echo $$ > pid.txt
exec $jre  -jar aserver.jar $*
```

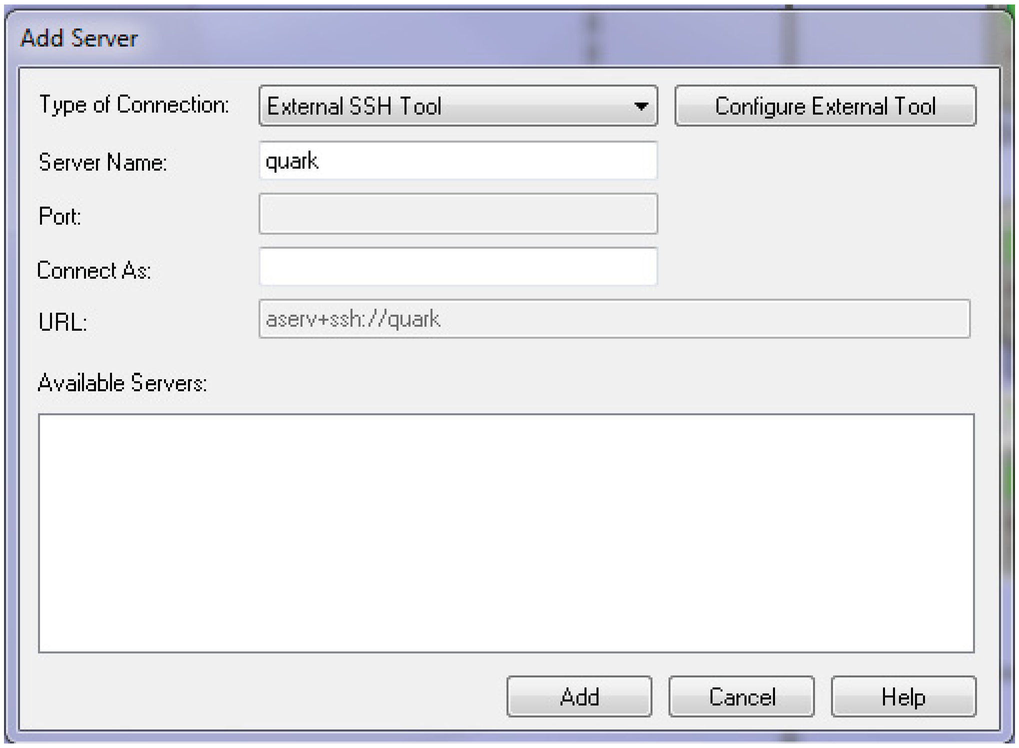
The two “export” commands are essentially the same as “module load starccm+/8.02.008.” They will ensure that version 8.02.008 of STAR-CCM+ is used.

Now go to /projects/USU/melter/Analysis/PhoenixInt/AServer7/analyses/melter.

This folder contains a folder called “MC Stuff 2” that contains all the files associated with the STAR-CCM+ component created before. If you want to change any of these files, use WinSCP or a similar file transfer program to transfer files from your desktop over to the “MC Stuff 2” folder. Make sure that in the scriptWrapper file used for HPC, the line that sets a run directory for the wrapper is commented out or deleted. Also, go through the macro and make sure that you are saving the scene pictures in a directory you want (currently, there is a directory called /projects/USU/melter/STAR_Pics that contains scene pictures).

Now we will go to ModelCenter and create a component from the AnalysisServer on HPC.

In ModelCenter, in the Server Browser on the bottom-left pane, right click on any of the plus-signs or click on the “Add server” button (computer with green plus sign). In the pop-up window, for “Type of Connection” select “External SSH Tool.” Type in “quark” for server name.



Add Server

Type of Connection: External SSH Tool

Server Name: quark

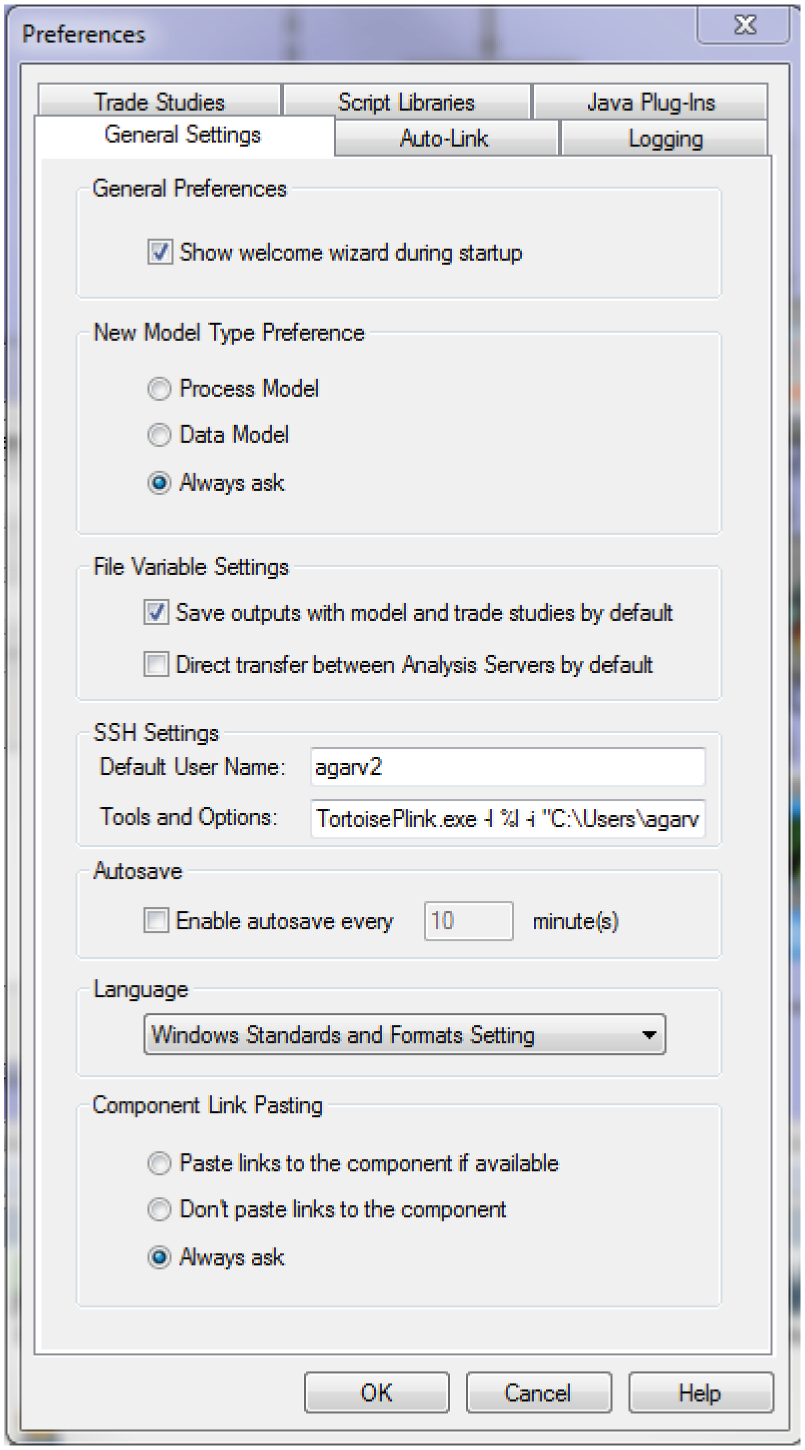
Port:

Connect As:

URL: aserv+ssh://quark

Available Servers:

Now click on “Configure External Tool.” Type in your username for Default User Name. The only thing you need to do for the “Tools and Options” box under SSH Settings is specify the absolute path to the aserver.sh file on the HPC (“/projects/USU/melter/Analysis/ASERVER/aserver.sh”).



General Preferences

[✓] Show welcome wizard during startup

New Model Type Preference

Process Model [ ] Data Model [ ] [● Always ask]

File Variable Settings

[✓] Save outputs with model and trade studies by default

[ ] Direct transfer between Analysis Servers by default

SSH Settings

Default User Name: agarv2

Tools and Options: TortoisePlink.exe -l %l -i "C:\Users\agarv

Autosave

[ ] Enable autosave every 10 minute(s)

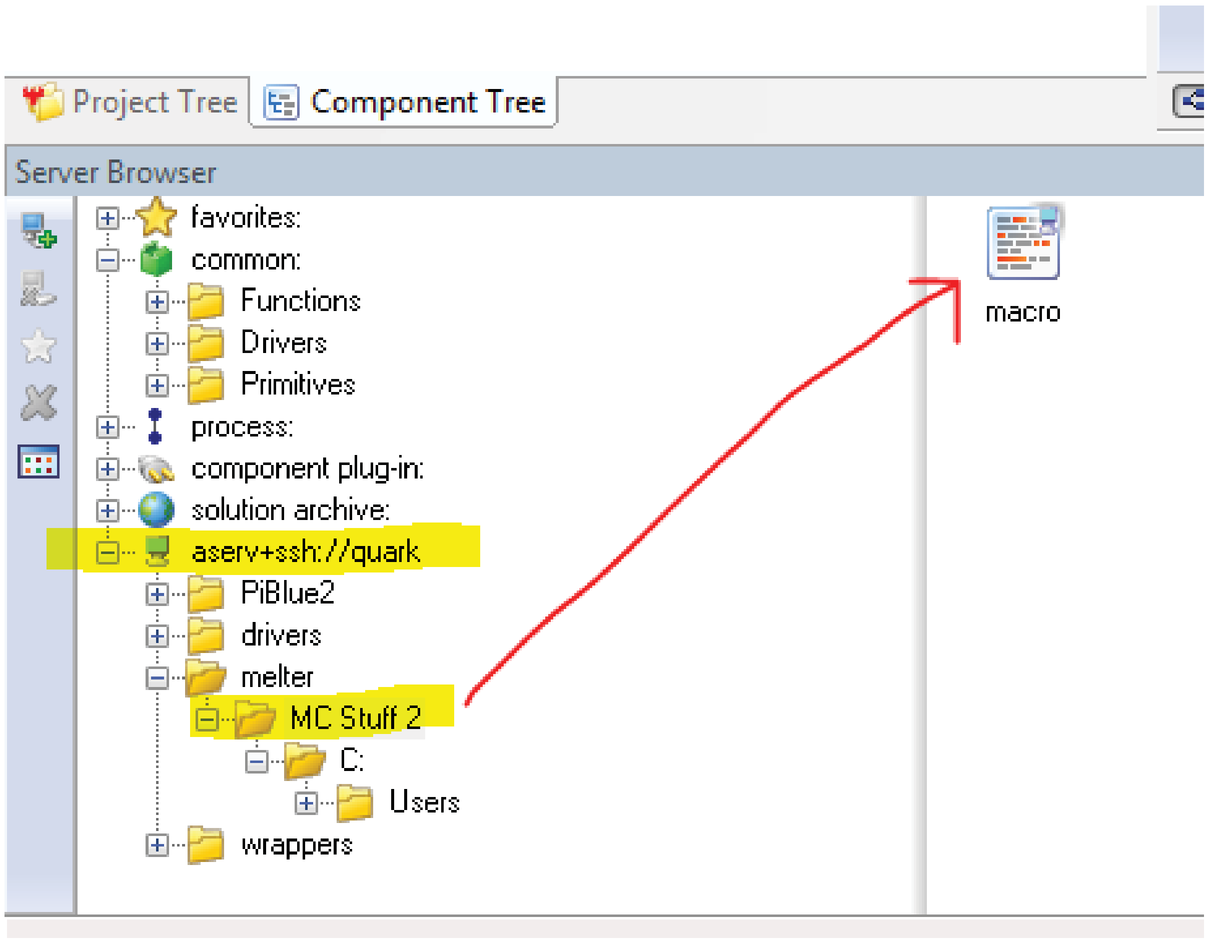
Language

Windows Standards and Formats Setting

Component Link Pasting

Paste links to the component if available [ ] Don't paste links to the component [ ] [● Always ask]

Go ahead and leave Tools and Options to whatever the Default is on your computer. Public and private keys generated by PuTTYGen can be used so that the user does not have to enter a password every time to connect to the HPC server.



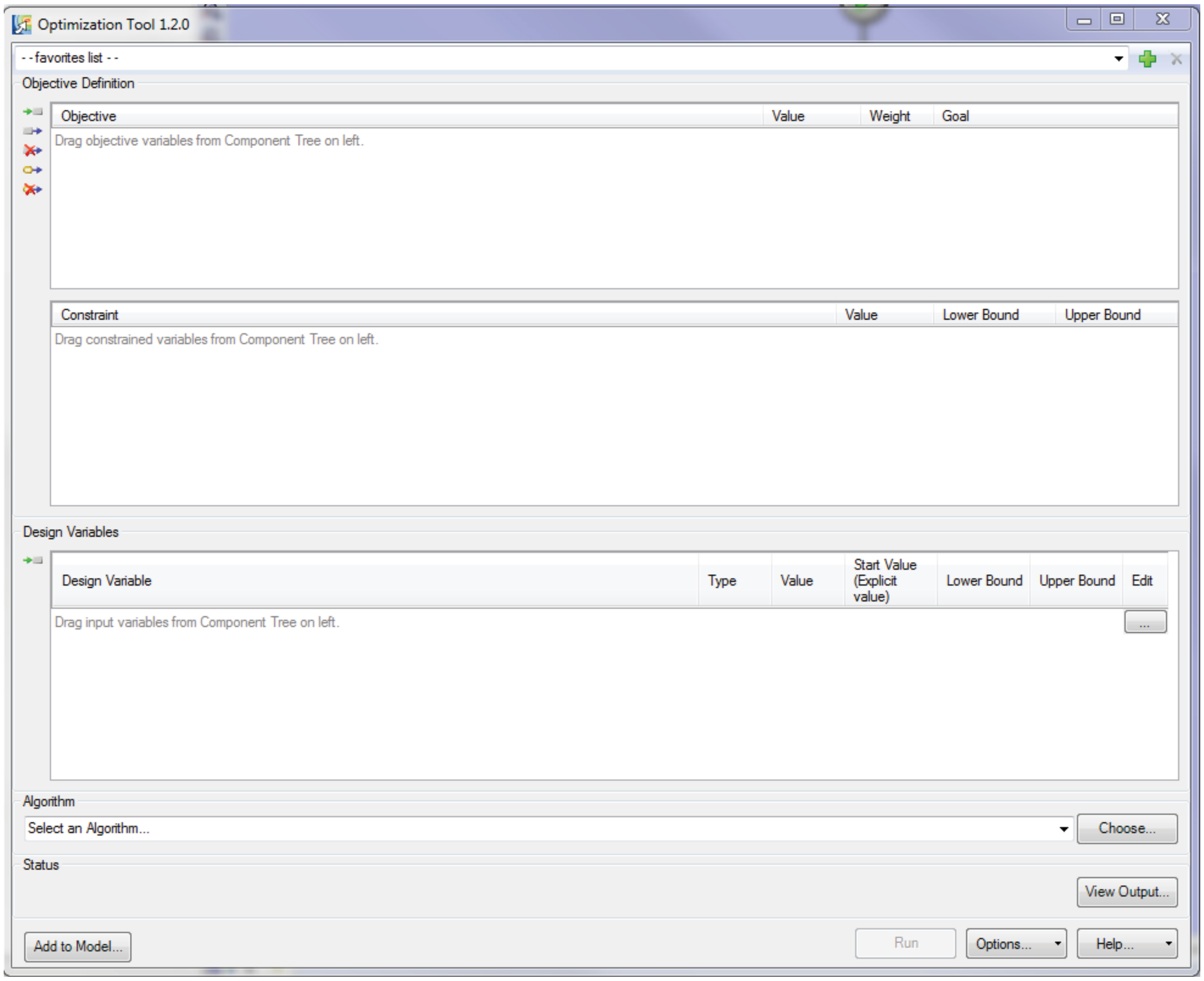
After clicking "OK," click on "Add" once back in the "Add Server" dialog.

Now you should be able to see an "aserv+ssh://quark" in your Server Browser. Upon expanding this server's node, you should see folders such as PiBlue2, drivers, wrappers, and most importantly, melter. Navigate to MC Stuff 2 under melter, and a component for your STAR simulation should show up in the bottom right pane.

Drag and drop this component into the model. This component will replace the QuickWrap component that we had previously in the non-HPC version. You can set the links the same way before, and you should now be able to run the coupled model.

2.8 Optimization

ModelCenter has extensive optimization capabilities. By going to the Tools menu and selecting "Optimization Tool," the following window pops up:



For our project, we could create a variable for melting rate. We would then drag this variable into the "Objective" section and set the goal to be "Maximize." Under the "Constraint" section, we would have to specify an acceptable range for melting rate values by assigning lower and upper bounds. Design variables would be variables to control in performing the optimization – for instance, the mass flux of the slurry entering the melter (MFEED1) would be a possible design variable. After specifying variables, an algorithm will have to be selected from the many offered by ModelCenter. Following the specification of all the above information, the optimization should be able to be added to the model.

3. MATLAB MODEL OF COLD CAP

3.1 Changes Made to the Original 1D MATLAB Model

In order to be used in the ModelCenter coupling, several changes had to be made to the original 1D MATLAB model of the cold cap. These changes are listed below.

1. `main2D` turned into function that takes in `TB1`, `QU1`, `MFEED1` and returns `QB1`, `TT1`, `MGAS1`, `MGLASS1`. `TB1` is temperature right below cold cap, `QU1` is heat flux from plenum to slurry, `MFEED1` is mass flux of slurry feed to cold cap. `QB1` is heat flux from molten glass to cold cap bottom, `TT1` is temperature at top of cold cap, `MGAS1` is mass flux of gases evolved during melting from top of cold cap to plenum space, and `MGLASS1` is mass flux of molten glass from bottom of cold cap.

2. Several global variables added to files so that `main2D` can share these variables with other files in model

3. Temperatures turned into Kelvin from Celsius at beginning and end of model to facilitate coupling (STAR-CCM+ model uses Kelvin)

4. Mass fluxes added to model to facilitate coupling with STAR-CCM+. Cold cap now uses mass influx of slurry feed to calculate mass outflows of molten glass and gases that leave the cold cap. This code was added to `physical_data.m`. Note that the variable `jdry` was added to represent the mass flow of dry feed (slurry without the 52.2% water). Then, `jg` and `jdry` were used to determine `MGAS1` and `MGLASS1`.

5. In `can7.m`, the input value of `QU` is used to calculate `QB`. This makes use of ΔH value needed to evaporate water and then heat the vapor to the temperature of the plenum space. ΔH values as well as the heat flux to melt the dry batch (33.8 kW/m^2) have been pulled from the PNL report that accompanied the cold cap model (http://www.pnl.gov/main/publications/external/technical_reports/PNNL-20278.pdf).

6. The rest of the files in the model are also functions.

   The MATLAB files used in the coupled model can be found in the next few pages.

3.2 Modified MATLAB Code

main2D.m

```
% Main2D.m ********************************************************************
%
% Runs the simulation. Main2D was originally
% a script, but has been changed to a function that takes in TB, QU, and
% MIN and returns QB, TT, MGAS and MGLASS. Main2D also calls the ODE solver
% that solves the heat balance ODE for the cold cap.
%
% Modified by INL, August 2013
%*****************************************************************************

function [QB1, TT1, MGAS1, MGLASS1] = main2D(TB1, QU1, MFEED1)
global time yyy T  QB TT TB QU MGAS MGLASS MFEED
TB = TB1-273.15; % convert to C, since STAR passes in K, not C
QU = QU1;
MFEED = MFEED1;
Thick=0.06; %cold cap thickness
Rad=0.1; %cold cap radius
M=100; % Number of finite volumes (axially)
N=2; % Number of finite volumes (radially)
% M=param.M; % Number of finite volumes (axially)
% N=param.N; % Number of finite volumes (radially)
% js0=0.0222*0.8; % mass flow to cold cap (based on 1D paper for now)

global hr qz qjz qz0 qjz0; % variables from function can7, which are necessary
globally
yini = init_cond; % initial guessed temperature profile - linear

time = 0:.0002:.01; % 0:60:3600
OP=odeset('NormControl','on','RelTol',1e-2,'AbsTol',1e-2,'MaxStep',6); %error
tolerances, maximum step size
[tt, yy] = ode15s(@(t,y) can7(t,y), time, yini, OP);

yyy=v2array3D(yy); % simulation results yy into 3D matrix
% 1st index = axial (z)
% 2nd index = radial (r)
% 3rd index = time (t)

%========================================================
%       postprocessing (results)
%========================================================
T=yyy(:,:,end); % last time
TT=T(1,1); %param.TT; % slurry temperature
% TB=T(end, end); %param.TB; % bottom temperature
physical_data % distribution of final cold cap properties
QB1 = QB;
TT1 = TT + 273.15; % Convert to K, since STAR accepts K, not C
MGAS1 = MGAS;
MGLASS1 = MGLASS;
```

init cond.m

```
% init_cond.m *************************************************************
%
% Sets up the initial temperature profile of the cold cap, which is passed into
% the ODE solver.
%
% Modified by INL, August 2013
%*************************************************************************

function yini = init_cond
global  TT TB
TT=100;   %cc top temperature
% param.TB=1100;  %cc bottom temperature
M = 100;
N = 2;
index=0;
Tini=ones(M*N,1); % initial conditions - linear temperature profile
for k=1:N
    for l=1:M
        index=index+1;
        Tini(index)=TT+(TB-TT)/M*l;
    end
end

yini=Tini;
end
```

physical data.m

```
% physical_data.m*******************************************************
%
% Solves for material properties (heat conductivities, heat capacities and
% spatial densities) using the temperature profile of the cold cap. This
% file also determines mass flows of the gaseous and solid phases
% in the cold cap.
%
% Modified by INL, August 2013
%**********************************************************************

function physical_data

global TT TB T lam lamT lamB js jg cpc cpg rho MFEED MGAS MGLASS
M = 100;
N = 2;

%-----------------------------------------------------
%       heat conductivity
%-----------------------------------------------------
lambdaeff = zeros(N,M); % thermal conductivity
for k=1:N
    for l=1:M
        if ((T(k,l)) < 727) %830 instead of 1500
        % ---Glass service report
        lambdaeff(k,l) = 0.06571 + 0.002114*(T(k,l)+273.15);
        % ---Petr Schill paper
        %      lambdaeff(i) =  0.5*exp(0.00233*(T(i,1)+273.15-290));
        elseif ((T(k,l)) < 800)
        lambdaeff(k,l) = -4.2007 + 0.0063807*(T(k,l)+273.15);
        else   % foam layer at T>800C - lambda=lambda/2
        lambdaeff(k,l) = (-4.2007 + 0.0063807*(T(k,l)+273.15))/2;
        end
    end
end
lamT= 0.06571 + 0.002114*(TT+273.15);
lamB=(-4.2007 + 0.0063807*(TB+273.15))/2;
lam=lambdaeff;

%------------------------------------------------------
%       mass flow - solid and gaseous phase
%------------------------------------------------------
%valid only in case of 1D (vertical) flow - no side flow
alpha = zeros(N,M); % based on TGA data
js = zeros(N,M); % mass flow of solid phase
jg = zeros(N,M); % mass flow of gas phase

% coefficients from heat conductivity paper (fitted to A0 data at 5 K/min)
a0 = 0.905;
a1 = 0.086;
aT0 = 561.8; %K
aT1 = 91.35; %K

alpha = a0 - a1.*atan(((T+273.15)-aT0)./aT1); % degree of conversion; ai is
fraction of material reacted in ith reaction
```

```
jdry = MFEED*0.478; % dry batch mass flow rate, based on 52.2% mass of water in
slurry
js = jdry*alpha;
jg = jdry*(alpha-alpha(N,M));
MGAS = jg(1,1);
% MGLASS = js(N,M);
MGLASS = jdry - MGAS;

%--------------------------------------------------------
%       heat capacity of condensed and gas phase
%--------------------------------------------------------
% cp of condensed phase - based on DSC data (Jaehun)
%                        - used also in the HC paper
cpc = zeros(N,M); % heat capacity condensed phase
for k=1:N
    for l=1:M
    if ((T(k,l)) < 600)
        cpc(k,l) = 0.272720463 - 0.23842748*(T(k,l)/1000) + 0.25363843*exp(-
((T(k,l)/1000-0.243943)^2/0.0010336)) + 0.500527*exp(-((T(k,l)/1000-
0.275096)^2/0.000188)) + 0.093806*exp(-((T(k,l)/1000-0.31286)^2/0.000323)) +
0.184088*exp(-((T(k,l)/1000-0.393729)^2/0.000697)) + 0.123335*exp(-
((T(k,l)/1000-0.4718234)^2/0.00204));
        cpc(k,l) = cpc(k,l) * 10000; % in J/(kg K)
    else
        cpc(k,l) = 1320; % constant heat capacity for T>600°C considered (1120
original, 1320 fitted)
    end
    end
end

% gas phase
cpg = zeros(N,M); % heat capacity gas
for k=1:N
    for l=1:M
    cpg(k,l) = 1003+0.21*(T(k,l)+273.15)-1.93e7/(T(k,l)+273.15)^2; %from
literature, Schill
    end
end

%--------------------------------------------------------
%       spatial density
%--------------------------------------------------------
% the same as in 1D model - based on foaming curves and TGA
% not needed for the calculation of temperature profile
%           - the temp profile is affected by mass flow js and jg
%           - the density only determines the velocity: j=v*rho (mass flow =
velocity x density)

rho = zeros(N,M); % spatial density
for k=1:N
    for l=1:M

        if ((T(k,l)) < 680)   % foam layer temperature boundary
        rho(k,l) = 970*alpha(k,l)/(-0.0000001*T(k,l)^2+0.00002*T(k,l)+1.001);
        elseif((T(k,l)) < 775)
        a00=245.82696587;
        a11=-1041.06249697;
        a22=1475.92823238;
        a33=-697.84383143;
        a44=0.00000000;
        rho(k,l) = 970*alpha(k,l)/(a44*(T(k,l)/1000)^4+a33*(T(k,l)/1000)^3 ...
            +a22*(T(k,l)/1000)^2+a11*(T(k,l)/1000)+a00);
        elseif((T(k,l)) < 960)
        a00=19.69000;
        a11=-47.72000;
        a22=29.88000;
        a33=0.00000;
        a44=0.00000;

        rho(k,l) = 970*alpha(k,l)/(a44*(T(k,l)/1000)^4+a33*(T(k,l)/1000)^3 ...
            +a22*(T(k,l)/1000)^2+a11*(T(k,l)/1000)+a00);
        else
        rho(k,l) = 541;
        end

    end
end

end
```

can7.m

```
% can7.m *********************************************************************
%
% Aids in the solution of the heat balance ODE. The output of the
% can7 function is used to determine the value of dT/dt at different values
% of T (temperature) and t (time), which is directly used by the ODE solver
% to obtain a solution.
%
% Modified by INL, August 2013
%****************************************************************************

function [dydt] = can7(time,yvec)
%CAN7 - model equations (heat balance)
%   state variables: 2D-temperatures in vector
global T lam lamT lamB js jg cpc cpg rho
global hr qz qjz qz0 qjz0 % variables needed globally
global QB TT TB QU MGAS MFEED MGLASS

%geometry
Len=0.06; %cold cap thickness
Rad=0.1; %cold cap radius
M=100; % Number of finite volumes (axially)
N=2; % Number of finite volumes (radially)
% js0=0.0222*0.8; % mass flow to cold cap (based on 1D paper for now)
hr = Rad/(N);    % dimensions of finite volume radially
hz = Len/(M);    % dimensions of finite volume axially


%rho=1300;   % density - need to change
%vel=4e-5;   % velocity not used now - will be result

% TT=100; % slurry temperature
%TB=param.TB; % bottom temperature - determined by STAR model

% unpack state variables (temperature vector-->matrix)
T=zeros(N,M); % first index=radially, second index=axially
cpc=zeros(N,M);
index=0;
for k=1:N
    for l=1:M
        index=index+1;
        T(k,l)=yvec(index);
    end
end

physical_data


% Calculation of volume of each finite volume (FV) and the axial and radial
% areas between FV
r=hr*(1:N);

V=zeros(1,N);
V(1)=pi*r(1)*r(1)*hz;
for k=2:N
```

```
    V(k)=pi*hz*(r(k)*r(k)-r(k-1)*r(k-1));
end

AR=zeros(1,N); % areas between FV radially
for k=1:N
    AR(k)=2*pi*hz*r(k);
end

AZ=zeros(1,N); % areas between FV axially
AZ(1)=pi*r(1)*r(1);
for k=2:N
    AZ(k)=pi*(r(k)*r(k)-r(k-1)*r(k-1));
end

% calculation of heat flows - radially
qr=zeros(N,M);
for l=1:M
    qr(1,l)=-(2*lam(2,l)*lam(1,l)/(lam(2,l)+lam(1,l)))*(T(2,l)-T(1,l))/(1.5*hr);
% f
    for k=2:(N-1)
        qr(k,l)=-(2*lam(k+1,l)*lam(k,l)/(lam(k+1,l)+lam(k,l)))*(T(k+1,l)-
T(k,l))/hr;
    end
%    qr(N,l)=-lam*(Tsurf-T(N,l))/(0.5*hr);  % flow of external KO into the area
    qr(N,l)=0; % radial heat flow not considered yet (1D)
end
qjr=zeros(N,M); %radial velocity flow is neglected in 1D

% calculation of heat flows - axially
qz=zeros(N,M);
qjz=zeros(N,M);
qz0=zeros(1,N); % qz(k,0) matlab cannot use 0 in index
qjz0=zeros(1,N); % qjz(k,0) matlab cannot use 0 in index
for k=1:N
    for l=1:M-1
        qz(k,l)=-(2*lam(k,l+1)*lam(k,l)/(lam(k,l+1)+lam(k,l)))*(T(k,l+1)-
T(k,l))/hz;
        qjz(k,l)=(js(k,l)*cpc(k,l)-jg(k,l)*cpg(k,l))*(T(k,l+1)+T(k,l))/2;
    end
    qz(k,M)= -(2*lamB*lam(k,M)/(lamB+lam(k,M)))*(TB-T(k,M))/(0.5*hz); % bottom
heat flow; TB (taken from melter model) used to get this value
    qjz(k,M)=(js(k,M)*cpc(k,M)-jg(k,M)*cpg(k,M))*(TB+T(k,M))/2; % bottom
velocity flow

    qz0(k) = -(2*lam(k,1)*lamT/(lam(k,1)+lamT))*(T(k,1)-TT)/(0.5*hz); % top heat
flow - QT
    qjz0(k) =(js(k,1)*cpc(k,1)-jg(k,1)*cpg(k,1))*(T(k,1)+TT)/2; % top velocity
flow
%%%%%%% Based on PNL paper: QB = QT + 33.8; QT = Htotal - QUl %%%%%%%%%%%%%%%
% Htotal = heat flux to evaporate water and then heat the vapor to the
% temperature of the plenum space; takes into account heat flow from free
% surface of molten glass to plenum space
%
% 33.8 kW/m2 = heat flux to melt dry batch to molten glass (from DSC data)
%
    jdry = 0.478*MFEED;
    jwat = 0.522 * MFEED;
    Hvap = jwat * 2.26e6;
    Hpre = (jwat*4185 + jdry*1300)*(100-30);
    Htotal = Hvap + Hpre;
    QB = Htotal - QU + (33.8e3);
    qz(k,M) = QB;
    qz0(k) = Htotal - QU; %  qz0 = QT
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end

% -----------------------------------
% heat balance of each finite volume
% -----------------------------------
dTdt=zeros(N,M);

% a) inner (leftmost) FV - cylinder
k=1;
    l=1;
    dTdt(k,l)=(-AR(k)*qr(k,l)+AZ(k)*(qz0(k)+qjz0(k))-AZ(k)*(qz(k,l)+qjz(k,l))) /
(V(k)*rho(k,l)*cpc(k,l));
    for l=2:M
        dTdt(k,l)=(-AR(k)*qr(k,l)+AZ(k)*(qz(k,l-1)+qjz(k,l-1))-
AZ(k)*(qz(k,l)+qjz(k,l))) / (V(k)*rho(k,l)*cpc(k,l));
    end
%end k=1

% b) other FV ("rings")
for k=2:N
    l=1;
    dTdt(k,l)=(AR(k-1)*qr(k-1,l)-AR(k)*qr(k,l)+AZ(k)*(qz0(k)+qjz0(k))-
AZ(k)*(qz(k,l)+qjz(k,l))) / (V(k)*rho(k,l)*cpc(k,l));
    for l=2:M
        dTdt(k,l)=(AR(k-1)*qr(k-1,l)-AR(k)*qr(k,l)+AZ(k)*(qz(k,l-1)+qjz(k,l-1))-
AZ(k)*(qz(k,l)+qjz(k,l))) / (V(k)*rho(k,l)*cpc(k,l));
    end
end

% pack state variables (matrix --> vector)
dydt=zeros(N*M,1); % column vector for integration
index=0;
for k=1:N
    for l=1:M
        index=index+1;
        dydt(index)=dTdt(k,l);
    end
end
```

v2array3D.m

```
% v2array3D.m ***************************************************************
%
% Compresses the solution from the ODE solver into a 3D
% matrix that represents the temperature solution for the cold cap at
% different times.
%
% Modified by INL, August 2013
%****************************************************************************

function [yout] = v2array3D(yin)
%V2M Vector (1D) to Array (3D)
%   yin is vector
%   yout is 3d-array
M=100;
N=2;
yout = zeros(N,M,51);
dim=size(yin); % first number is the number of rows (times), the second number
is the number of columns (temperature)

for t=1:dim(1); % for all time steps

    yout(:,:,t)=zeros(N,M);
    index=0;
    for k=1:N
        for l=1:M
            index=index+1;
            yout(k,l,t)=yin(t,index);
        end
    end
end
end
```

3.3 Data Dictionary for Matlab Cold Cap Model

Variables
`M` – number of finite volumes axially
`N` – number of finite volumes radially

`alpha` – NxM matrix containing mass fraction of condensed phase
`AR` – 1xN vector containing areas between finite volumes in radial direction
`AZ` – 1xN vector containing areas between finite volumes in axial direction
`cpc` – Heat capacity of condensed phase; NxM array
`cpg` – Heat capacity of gas phase; NxM array
`dTdt` – NxM array containing solution to right hand side of heat balance ODE (value of dT/dt)
`dydt –` (N*M)x1 column vector containing values from `dTdt`
`Hpre` – Heat flux needed to preheat slurry from feed temperature to temperature of boiling slurry
`hr` – dimension of a single finite volume in radial direction
`Htotal` – Sum of `Hvap` and `Hpre`
`Hvap` – Heat flux needed to evaporate water
`hz` – dimension of a single finite volume in axial direction
`jdry` – Mass flux of dry batch to cold cap (based on 52.2% water content in slurry)
`jg` – Mass flux of gas phase; NxM array
`js` – Mass flux of solid phase; NxM array
`jwat` – Mass flux of water to cold cap in slurry (based on 52.2% water content in slurry)
`lam` – same as `lambdaeff`
`lamB` – thermal conductivity at bottom of cold cap
`lambdaeff` – NxM array containing thermal conductivity values
`lamT` – thermal conductivity at top of cold cap
`Len` – cold cap thickness
`MGAS` – Mass flux of gases evolved from cold cap
`MGLASS` – Mass flux of molten glass from cold cap
`MFEED` – Mass flux of slurry to cold cap
`QB` – Heat flux from molten glass to bottom of cold cap
`qjr` – NxM array containing radial velocity flow values
`qjz` – NxM array containing axial velocity flow values
`qjz0` – 1xN vector that represents velocity flow at top of cold cap
`qr` – NxM array containing radial heat flow values
`QU` – Heat flux from plenum space to slurry
`qz` – NxM array containing axial heat flow values
`qz0` – 1xN vector that represents heat flow at top of cold cap
`Rad` – cold cap radius
`rho` – Spatial density; NxM array
`T` (in can7.m) – Temperature profile of cold cap in the form of an NxM array
`T` (in main2D.m) - Final temperature solution at the last time value
`TB` – Temperature at bottom of cold cap
`Thick` – cold cap thickness
`time` – Row vector of time values, in seconds, going from 0 to 3600 in steps of 60
`tt` – Column vector with same time values as the variable `time`

`TT` – Temperature at top of cold cap
`V` – 1xN vector containing volume of each finite element
`yini` – linear temperature profile going from 110°C to 1100°C
`yy` – Solution array in which each row corresponds to temperature solution at the time in the corresponding row of `tt`
`yyy` – Solution array, in 3D form; first dimension is axial, second is radial and third is time

3.4 Equations used for Material Properties (in physical_data.m)

3.4.1 Heat Conductivity

For T < 727 K:

$$\mathrm{lambdaeff(T)} = 0.06571 + 0.002114(T + 273.15)$$

For T ≥ 727 K and T < 800 K:

$$\mathrm{lambdaeff(T)} = -4.2007 + 0.0063807(T + 273.15)$$

For T ≥ 800 K (foam layer heat conductivity is approximated as half that of region just above foam layer):

$$\mathrm{lambdaeff(T)} = [-4.2007 + 0.0063807(\mathrm{T} + 273.15)]/2$$

3.4.2 Degree of Conversion (alpha) – based on TGA data

a0 = 0.905
a1 = 0.086
aT0 = 561.8
aT1 = 91.35

$$\mathrm{alpha(T)} = \mathrm{a0} * \mathrm{atan}\left(\frac{(\mathrm{T} + 273.15) - \mathrm{aT0}}{\mathrm{aT1}}\right)$$

3.4.3 Mass Flows

1. Mass flow of solid phase (js) with units of kg/m^{2}s

   $$\mathrm{js} = \mathrm{jdry} * \mathrm{alpha}$$

2. Mass flow of gas phase (jg) with units of kg/m^{2}s

   $$\mathrm{js} = \mathrm{jdry} * (\mathrm{alpha} - \mathrm{alpha}_{\mathrm{bottomOfColdCap}})$$

In both cases, jdry = MFEED*0.478, where MFEED is the mass flux of the slurry feed. Mass flux of gases from cold cap top (1st row and 1st column of `jg` matrix)

$$\mathrm{MGAS} = \mathrm{jg}(1,1)$$

3. Mass flux of molten glass from cold cap bottom – conservation of mass used

   $$\mathrm{MGLASS} = \mathrm{jdry} - \mathrm{MGA}$$

Units on jdry, js, jg, MGAS and MGLASS are kg/m^{2}s.

3.4.4 Heat Capacities

1. Heat capacity of gas phase (cpg) approximated by cpg for carbon dioxide:

   $$\mathrm{cpg(T)} = 1003 + 0.21\mathrm{T} - \frac{1.93\mathrm{x}10^7}{\mathrm{T}^2}$$

   Units: cpg in J kg^{-1}K^{-1}, T in K and T≥373 K

2. Heat capacity of condensed phase (cpc)

For T < 600K: $$\mathrm{cpc}(T) = 1000\left[0.272720463 - 0.23842748\left(\frac{T}{1000}\right) + 0.25363843e^{-\left(\frac{\left(\frac{T}{1000}-0.243943\right)^2}{0.0010336}\right)} + 0.500527e^{-\left(\frac{\left(\frac{T}{1000}-0.275096\right)^2}{0.000188}\right)} + 0.093806e^{-\left(\frac{\left(\frac{T}{1000}-0.31286\right)^2}{0.000323}\right)} + 0.184088e^{-\left(\frac{\left(\frac{T}{1000}-0.393729\right)^2}{0.000697}\right)} + 0.123335e^{-\left(\frac{\left(\frac{T}{1000}-0.4718234\right)^2}{0.00204}\right)}\right]$$

For T ≥ 600K: cpc = 1320

3.4.5 Spatial Density

For T < 680K:

$$\mathrm{rho}(T) = 970 * \frac{\mathrm{alpha}(T)}{-0.0000001T^2 + 0.0002T + 1.001}$$

For T ≥ 680K and T < 775K:
a00=245.82696587
a11=-1041.06249697
a22=1475.92823238
a33=-697.84383143
a44=0.00000000

$$\mathrm{rho}(T) = 970 * \frac{\mathrm{alpha}(T)}{a44\left(\frac{T}{1000}\right)^4 + a33\left(\frac{T}{1000}\right)^3 + a22\left(\frac{T}{1000}\right)^2 + a11\left(\frac{T}{1000}\right) + a00}$$

For T ≥ 775 K and T < 960 K:
a00=19.69000
a11=-47.72000
a22=29.88000
a33=0.00000
a44=0.00000

$$\mathrm{rho}(T) = 970 * \frac{\mathrm{alpha}(T)}{a44\left(\frac{T}{1000}\right)^4 + a33\left(\frac{T}{1000}\right)^3 + a22\left(\frac{T}{1000}\right)^2 + a11\left(\frac{T}{1000}\right) + a00}$$

For T > 960 K: rho = 541

3.4.6 Calculation of QB using QU (in can7.m)

1. Mass flow of dry batch, based on slurry water content of 52.2% (mass percent):

$$\text{jdry} = 0.478 * \text{MFEED} \quad [\text{kg/m}^2\text{s}]$$

2. Mass flow of water from slurry, based on slurry water content of 52.2% (mass percent):

$$\text{jwat} = 0.522 * \text{MFEED} \quad [\text{kg/m}^2\text{s}]$$

3. Heat flux to evaporate water, using jwat and evaporation heat of water ($2.26\text{x}10^6$ W/m^2):

$$\text{Hvap} = \text{jwat} * 2.26\text{e}6 \quad [\text{W/m}^2]$$

4. Heat flux to preheat slurry from the feed temperature of 30°C to 100°C, the temperature of the boiling slurry; uses jwat, jdry, specific heat capacities of water and the dry batch, and difference between final and initial temperatures:

$$\text{Hpre} = (\text{jwat}*4185 + \text{jdry}*1300)*(100\text{-}30) \quad [\text{W/m}^2]$$

5. Total heat to preheat slurry and then to evaporate water from slurry:

$$\text{Htotal} = \text{Hvap} + \text{Hpre} \quad [\text{W/m}^2]$$

6. Calculation of QB (heat flux necessary from molten glass to cold cap bottom), using a value of 33.8 kW/m^2 for heat flux necessary to melt dry batch to molten glass:

$$\text{QB} = \text{Htotal - QU} + (33.8\text{e}3) \quad [\text{W/m}^2]$$

Slurry water content of 52.2%, initial and final feed temperatures, evaporation heat of water and heat necessary to melt dry batch all obtained from http://www.pnl.gov/main/publications/external/technical_reports/PNNL-20278.pdf (Section 5.4 titled "Water Evaporation").

3.5 Cold Cap Model Documentation

The 1D MATLAB model of the cold cap for a slurry-fed waste glass melter, developed by Pokorny, et. al. in [1], consists of the following 5 files:

1. main2D.m
2. init_cond.m
3. physical_data.m
4. can7.m
5. v2array3d.m

The main simulation is contained in the file main2D.m, which is the file that must be run from the MATLAB command window to create the simulation. This file uses the other 4 files as helpers to aid in the creation of the simulation. Below is a step-by-step explanation of the code in the main2D.m file as well as the code in the helper files. Most of the descriptions of the scientific purpose behind the code are pulled from [1].

Lines 1-15 of main2D.m are as follows:

```
function [QB1, TT1, MGAS1, MGLASS1] = main2D(TB1, QU1, MFEED1)
global time yyy T  QB TT TB QU MGAS MGLASS MFEED
TB = TB1-273.15; % convert to C, since STAR passes in K, not C
QU = QU1;
MFEED = MFEED1;
Thick=0.06; %cold cap thickness
Rad=0.1; %cold cap radius
M=100; % Number of finite volumes (axially)
N=2; % Number of finite volumes (radially)
% M=param.M; % Number of finite volumes (axially)
% N=param.N; % Number of finite volumes (radially)
% js0=0.0222*0.8; % mass flow to cold cap (based on 1D paper for now)

global hr qz qjz qz0 qjz0; % variables from function can7, which are
necessary globally
yini = init_cond; % initial guessed temperature profile - linear
```

Right away, it is important to note that this file is a function and not a script (this is a change made from the original model). Therefore, whatever variables are declared in this file will only exist within the scope of this file. In order for this file to share variables with other files, global variables will have to be used. This function has three inputs: TB1, the temperature of the molten glass right below the cold cap, QU1, the heat transfer from the plenum space to the slurry and MFEED1, the mass flux of the slurry feed to the cold cap. The function has four outputs: QB1, the heat transfer from the molten glass to the cold cap bottom, TT1, the temperature at the top of the cold cap, MGAS1, the mass flux of gases evolving from the cold cap and MGLASS1, the mass flux of the molten glass emerging from the cold cap bottom.

These lines of code declare variables that store the cold cap thickness (Thick) and radius (Rad) and also the number of finite volumes axially (M) and radially (N). Note that the input value of TB1 is converted into degrees C from degrees K when it is stored into the variable TB, since STAR-CCM+ produces temperatures in K while this model deals with temperature in degrees C.

The variables time, yyy, T, QB, TT, TB, QU, MGAS, MGLASS, MFEED, hr, qz, qjz, qz0, and qjz0 are declared to be global variables. This allows the main2D.m file to share the values of these variables with any other file that declares these variables as global.

A call is then made to init_cond.m, a helper file which will create an initial linear temperature profile for the cold cap. This file is discussed below.

Contents of init_cond.m:

```
function yini = init_cond
global  TT TB
TT=100;   %cc top temperature
% param.TB=1100;  %cc bottom temperature
M = 100;
N = 2;
index=0;
Tini=ones(M*N,1); % initial conditions - linear temperature profile
for k=1:N
    for l=1:M
        index=index+1;
        Tini(index)=TT+(TB-TT)/M*l;
    end
end
yini=Tini;
end
```

Two variables are declared to be global by this file – TT, the temperature at the top of the cold cap, and TB, the temperature at the bottom of the cold cap. Since main2D.m also declares these variables to be global, the two files can share the values of these variables (if one file changes the value of either of the variables, the value automatically changes for the other file as well). These two temperatures, 100°C and 1100°C for the top and bottom of the cold cap, respectively, are boundary conditions for temperature. Based on these temperatures, a linear temperature profile for the cold cap is then created using for loops and the boundary temperatures. This temperature profile, Tini, is initially represented as a column vector of 200 rows containing all ones. The appropriate temperature value at each index is then iteratively calculated based on the boundary temperatures. Thus, the values in Tini range from 110°C to 1100°C, first from indices 1-100, and then again from indices 101-200. This temperature profile is then also stored in the variable yini.

Once init_cond.m runs in its entirety, main2D.m continues to run where it left off. Below are lines 17-19 of main2D.m.

```
time = 0:.0002:.01; % 0:60:3600
OP=odeset('NormControl','on','RelTol',1e-2,'AbsTol',1e-2,'MaxStep',6);
[tt, yy] = ode15s(@(t,y) can7(t,y), time, yini, OP);
```

These lines perform integration of the heat balance ODE for the cold cap. A vector of times is first created, containing values from 0s to 0.01s in steps of 0.0002s ([0 0.0002 0.0004 0.0006… 0.01]). For the MATLAB to STAR-CCM+ coupling, this line of code needs to reflect the physical time and number of time steps taken in the STAR-CCM+ simulation so that the MATLAB and STAR-CCM+ models remain synchronized.

Next, the built-in MATLAB function odeset is used generate a structure, OP, which contains error tolerances and the maximum step size for the integration. Another built-in function, ode15s, is then used to perform the integration. This solver iteratively solves equations of the form y' = f(t,y). The solver must be provided with a way to calculate y' at different values of t and y; therefore, the first input to the ode15s function is a function handle to can7.m. A function handle is a way of accessing a function, and can7.m contains a function that outputs the value of y' (i.e. the right hand side of the heat balance ODE). The other inputs of ode15s include time, the time vector discussed previously, yini, the initial linear temperature profile, and OP, the structure containing information about error tolerances and the maximum step size. The outputs of ode15s are tt and yy: tt is a column vector of times, and yy is a solution matrix in which each row corresponds to the solution at the time in the corresponding row of tt.

As mentioned before, during the integration, can7.m is used to evaluate the value of y'. In this case, of course, y represents temperature and t represents time. Below are lines 1-15 of can7.m.

```
function [dydt] = can7(time,yvec)
%CAN7 - model equations (heat balance)
%   state variables: 2D-temperatures in vector
global T lam lamT lamB js jg cpc cpg rho
global hr qz qjz qz0 qjz0 % variables needed globally
global QB TT TB QU MGAS MFEED MGLASS

%geometry
Len=0.06; %cold cap thickness
Rad=0.1; %cold cap radius
M=100; % Number of finite volumes (axially)
N=2; % Number of finite volumes (radially)
% js0=0.0222*0.8; % mass flow to cold cap (based on 1D paper for now)
hr = Rad/(N);    % dimensions of finite volume radially
hz = Len/(M);    % dimensions of finite volume axially
```

Notice that this function shares the variables QB, TT, TB, QU, MGAS, MFEED, MGLASS, T, hr, qz, qjz, qz0, and qjz0 with main2D.m, as both files declare these variables as global. The cold cap thickness and radius and the number of finite volumes in the axial and radial directions are then defined. The dimensions of each FV axially and radially are then determined. The radial dimension (hr) is found by dividing the cold cap radius by the number of FV in the radial direction, and the axial dimension (hz) is found by dividing the cold cap thickness by the number of FV in the axial direction.

Lines 24 to 35 of can7.m are as follows:

```
% unpack state variables (temperature vector-->matrix)
T=zeros(N,M); % first index=radially, second index=axially
cpc=zeros(N,M);
index=0;
for k=1:N
    for l=1:M
        index=index+1;
        T(k,l)=yvec(index);
    end
end

physical_data
```

The above block of code uses nested for loops to convert the temperature vector that is passed into the function (yvec) into an NxM matrix called T (remember that N and M are the number of FV in the radial and axial directions, respectively). The first and second rows of T both contain temperature values that range from 110°C to 1100°C. Finally, a call is made to physical_data.m, which is a function that loads material properties into memory. This file is analyzed below.

The first section of physical_data.m is as follows:

```
function physical_data
global TT TB T lam lamT lamB js jg cpc cpg rho MFEED MGAS MGLASS
M = 100;
N = 2;
%------------------------------------------------------
%       heat conductivity
%------------------------------------------------------
lambdaeff = zeros(N,M); % thermal conductivity
for k=1:N
    for l=1:M
        if ((T(k,l)) < 727) %830 instead of 1500
        % ---Glass service report
        lambdaeff(k,l) = 0.06571 + 0.002114*(T(k,l)+273.15);
        % ---Petr Schill paper
        %       lambdaeff(i) =  0.5*exp(0.00233*(T(i,1)+273.15-290));
        elseif ((T(k,l)) < 800)
        lambdaeff(k,l) = -4.2007 + 0.0063807*(T(k,l)+273.15);
        else    % foam layer at T>800C - lambda=lambda/2
        lambdaeff(k,l) = (-4.2007 + 0.0063807*(T(k,l)+273.15))/2;
        end
    end
end
lamT= 0.06571 + 0.002114*(TT+273.15);
lamB=(-4.2007 + 0.0063807*(TB+273.15))/2;
lam=lambdaeff;
```

Note that physical_data.m is a function, just like main2D.m; therefore, it uses global variables to share data with the other files. The above section of code calculates the values of thermal conductivity using the temperature matrix (T) that was created by can7.m. These values are stored in an NxM matrix called lambdaeff, just like the matrix T. Initially, there is a 0 at each index in lambdaeff. Nested for loops are then used to iterate through T and use the temperature at each index to calculate the thermal conductivity at the same index in lambdaeff. The different values of temperature govern which equation is used to calculate the thermal conductivity, as the conductivity differs from region to region in the cold cap. Thus, in this code, the open porosity layer is treated differently from the foam layer. The first two equations (the equations for temperatures up to 800°C), used to calculate heat conductivity for the open porosity layer, are based on literature. The third equation governs the heat conductivity of the foam layer, which has not been calculated experimentally. Therefore, the thermal conductivity of the foam layer is approximated as half the heat conductivity of the region just above the foam layer.

The next section of physical_data.m concerns the mass flux values for the solid and gaseous phases:

```
%-------------------------------------------------------
%       mass flow - solid and gaseous phase
%-------------------------------------------------------
%valid only in case of 1D (vertical) flow - no side flow
alpha = zeros(N,M); % based on TGA data
js = zeros(N,M); % mass flow of solid phase
jg = zeros(N,M); % mass flow of gas phase

% coefficients from heat conductivity paper (fitted to A0 data at 5 K/min)
a0 = 0.905;
a1 = 0.086;
aT0 = 561.8; %K
aT1 = 91.35; %K

alpha = a0 - a1.*atan(((T+273.15)-aT0)./aT1); % degree of conversion; ai
is fraction of material reacted in ith reaction
jdry = MFEED*0.478; % dry batch mass flow rate, based on 52.2% mass of
water in slurry
js = jdry*alpha;
jg = jdry*(alpha-alpha(N,M));
MGAS = jg(1,1);
% MGLASS = js(N,M);
MGLASS = jdry - MGAS;
```

In this section, coefficients pulled from literature and the temperature matrix created in can7.m (T) are used to calculate the max flux values for the solid (js) and gaseous (jg) phases. Both js and jg initially are NxM matrices consisting of all zeros. The temperature matrix, T, is then used to calculate alpha, the degree of feed-to-glass conversion. The variables a0, a1, aT0 and aT1, which are also used to calculate alpha, are all based on thermal gravimetric analysis (TGA) data measured at a constant heating rate of 20 K/min. Notice that the mass flux of the dry batch, jdry, is then obtained from MFEED (the mass flux of slurry to the cold cap) based on a 52.2% mass water content. The value of jdry is then used in conjunction with alpha to calculate the mass fluxes of the solid and gaseous phases within the cold cap. The mass flux of the gas at the top of the cold cap is then assigned to the output variable MGAS, and the difference between the incoming dry batch mass flux, jdry, and MGAS is assigned to the variable MGLASS to ensure conservation of mass.

In this next section of physical_data.m, the heat capacities of the condensed and gas phases are calculated.

```
%-------------------------------------------------------
%       heat capacity of condensed and gas phase
%-------------------------------------------------------
% cp of condensed phase - based on DSC data (Jaehun)
%                        - used also in the HC paper
cpc = zeros(N,M); % heat capacity condensed phase
for k=1:N
    for l=1:M
    if ((T(k,l)) < 600)
        cpc(k,l) = 0.272720463 - 0.23842748*(T(k,l)/1000) +
0.25363843*exp(-((T(k,l)/1000-0.243943)^2/0.0010336)) + 0.500527*exp(-
((T(k,l)/1000-0.275096)^2/0.000188)) + 0.093806*exp(-((T(k,l)/1000-
0.31286)^2/0.000323)) + 0.184088*exp(-((T(k,l)/1000-0.393729)^2/0.000697))
+ 0.123335*exp(-((T(k,l)/1000-0.4718234)^2/0.00204));
        cpc(k,l) = cpc(k,l) * 10000; % in J/(kg K)
    else
        cpc(k,l) = 1320; % constant heat capacity for T>600°C considered
(1120 original, 1320 fitted)
    end
    end
end

% gas phase
cpg = zeros(N,M); % heat capacity gas
for k=1:N
    for l=1:M
    cpg(k,l) = 1003+0.21*(T(k,l)+273.15)-1.93e7/(T(k,l)+273.15)^2; %from
literature, Schill
    end
end
```

In this section, the heat capacity of the condensed phase (cpc) is calculated using the temperature matrix, T. Initially, cpc is an NxM array consisting of all zeros. Nested for loops are then used to iterate through T, calculate the value of the heat capacity for each temperature, and place that value in the corresponding index in the cpc matrix. For temperatures up to 600°C, the equation used for cpc is based on published differential scanning calorimetry (DSC) data. For temperatures above 600°C, the heat capacity is assumed to be a constant 1320 J/(kg-K). A technique similar to the one used to create the cpc matrix is used to create the matrix for the heat capacity of the gas phase (cpg). The equation used for calculating cpg is the equation for the heat capacity of carbon dioxide.

In the last section of physical_data.m, the spatial density matrix, rho, is calculated in a manner similar to that of the properties calculated in the previous sections. Calculations are based on foaming curves and the results of TGA. The different temperature regions of the cold cap have different values for spatial density; therefore, in this code, the temperature value is used to determine what equation is used to calculate the spatial density. Notice that for temperatures above 960°C, the density is treated as a constant.

```
%-------------------------------------------------------
%       spatial density
%-------------------------------------------------------
% the same as in 1D model - based on foaming curves and TGA
% not needed for the calculation of temperature profile
%            - the temp profile is affected by mass flow js and jg
%            - the density only determines the velocity: j=v*rho (mass flow =
velocity x density)

rho = zeros(N,M); % spatial density
for k=1:N
    for l=1:M

        if ((T(k,l)) < 680)    % foam layer temperature boundary
        rho(k,l) = 970*alpha(k,l)/(-0.0000001*T(k,l)^2+0.00002*T(k,l)+1.001);
        elseif((T(k,l)) < 775)
        a00=245.82696587;
        a11=-1041.06249697;
        a22=1475.92823238;
        a33=-697.84383143;
        a44=0.00000000;
        rho(k,l) = 970*alpha(k,l)/(a44*(T(k,l)/1000)^4+a33*(T(k,l)/1000)^3 ...
            +a22*(T(k,l)/1000)^2+a11*(T(k,l)/1000)+a00);
        elseif((T(k,l)) < 960)
        a00=19.69000;
        a11=-47.72000;
        a22=29.88000;
        a33=0.00000;
        a44=0.00000;

        rho(k,l) = 970*alpha(k,l)/(a44*(T(k,l)/1000)^4+a33*(T(k,l)/1000)^3 ...
            +a22*(T(k,l)/1000)^2+a11*(T(k,l)/1000)+a00);
        else
        rho(k,l) = 541;
        end

    end
end
```

Because physical_data.m is now finished executing, MATLAB returns to can7.m. Shown below are lines 38 to 57 of can7.m:

```
% calculation of area and volume of each finite volume (FV)
r=hr*(1:N);

V=zeros(1,N);
V(1)=pi*r(1)*r(1)*hz;
for k=2:N
    V(k)=pi*hz*(r(k)*r(k)-r(k-1)*r(k-1));
end

AR=zeros(1,N); % areas between FV v radially
for k=1:N
    AR(k)=2*pi*hz*r(k);
end

AZ=zeros(1,N); % areas between FV v axially
AZ(1)=pi*r(1)*r(1);
for k=2:N
    AZ(k)=pi*(r(k)*r(k)-r(k-1)*r(k-1));
end
```

In this section, volumes of FV and areas between FV are calculated. The FV themselves consist of an inner cylinder surrounded by cylindrical shells. Therefore, the formula for the volume of a cylinder is used to find the volume of each FV. The area between FV in the radial direction uses the formula for the surface area of a cylinder, and the area between FV in the axial direction uses the formula for the area of a circle. All of these computations use the axial and radial dimensions (hr and hz, respectively) of each FV that were calculated before.

The next section of can7.m, lines 59 to 102, is shown below.

```
% calculation of heat flows - radially
qr=zeros(N,M);
for l=1:M
    qr(1,l)=-(2*lam(2,l)*lam(1,l)/(lam(2,l)+lam(1,l)))*(T(2,l)-T(1,l))/(1.5*hr); % f
    for k=2:(N-1)
        qr(k,l)=-(2*lam(k+1,l)*lam(k,l)/(lam(k+1,l)+lam(k,l)))*(T(k+1,l)-T(k,l))/hr;
    end
%    qr(N,l)=-lam*(Tsurf-T(N,l))/(0.5*hr);  % flow of external KO into the area
    qr(N,l)=0; % radial heat flow not considered yet (1D)
end
qjr=zeros(N,M); %radial velocity flow is neglected in 1D

% calculation of heat flows - axially
qz=zeros(N,M);
qjz=zeros(N,M);
qz0=zeros(1,N); % qz(k,0) matlab cannot use 0 in index
qjz0=zeros(1,N); % qjz(k,0) matlab cannot use 0 in index
for k=1:N
    for l=1:M-1
        qz(k,l)=-(2*lam(k,l+1)*lam(k,l)/(lam(k,l+1)+lam(k,l)))*(T(k,l+1)-T(k,l))/hz;
        qjz(k,l)=(js(k,l)*cpc(k,l)-jg(k,l)*cpg(k,l))*(T(k,l+1)+T(k,l))/2;
    end
    qz(k,M)= -(2*lamB*lam(k,M)/(lamB+lam(k,M)))*(TB-T(k,M))/(0.5*hz); % bottom heat flow;
TB (taken from melter model) used to get this value
    qjz(k,M)=(js(k,M)*cpc(k,M)-jg(k,M)*cpg(k,M))*(TB+T(k,M))/2; % bottom velocity flow

    qz0(k) = -(2*lam(k,1)*lamT/(lam(k,1)+lamT))*(T(k,1)-TT)/(0.5*hz); % top heat flow - QT
    qjz0(k) =(js(k,1)*cpc(k,1)-jg(k,1)*cpg(k,1))*(T(k,1)+TT)/2; % top velocity flow
    %%%%%%% Based on PNL paper: QB = QT + 33.8; QT = Htotal - QUl %%%%%%%%%%%%%%
    % Htotal = heat flux to evaporate water and then heat the vapor to the
    % temperature of the plenum space; takes into account heat flow from free
    % surface of molten glass to plenum space
    %
    % 33.8 kW/m2 = heat flux to melt dry batch to molten glass (from DSC data)
    %
        jdry = 0.478*MFEED;
        jwat = 0.522 * MFEED;
        Hvap = jwat * 2.26e6;
        Hpre = (jwat*4185 + jdry*1300)*(100-30);
        Htotal = Hvap + Hpre;
        QB = Htotal - QU + (33.8e3);
        qz(k,M) = QB;
        qz0(k) = Htotal - QU; %  qz0 = QT
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
end
```

In this section, the radial and axial heat and velocity flows (qr, qz, qjr, and qjz, respectively) are calculated. Because this model is 1D, the radial velocity flow matrix contains all zeros, and the second row of the radial heat flow matrix also contains zeros. The axial heat and velocity flows, however, must be included in the model, so they are calculated using the appropriate equations (which take into consideration material properties such as thermal conductivity and heat capacities as well as FV dimensions). The heat flow equations are based on Fourier's law.

After heat and velocity flows are calculated, the value of QU (which is an input from the STAR-CCM+ simulation) is used to calculate QB, the heat transfer from the molten glass to the cold cap bottom. The variables jdry (mass flux of dry batch) and jwat (mass flux of water from slurry) are first calculated using MFEED and a slurry water content of 52.2%. The heat flux to evaporate water, Hvap, is then calculated using jwat and the evaporation heat of water (2.26×10^6 W/m^2). The heat to preheat the slurry from the feed temperature of 30°C to 100°C, the temperature of the boiling slurry, is calculated as Hpre using the jwat, jdry, the specific heat capacities of water and the dry batch, and the difference between the final and initial temperatures. The sum of Hvap and Hpre is stored as Htotal, which is then used in conjunction with QU to find QB. QU is subtracted from Htotal to find the remaining heat that needs to be supplied from the molten glass to the cold cap to aid in the preheating and drying of the slurry, and then the heat necessary to melt the dry batch to molten glass, which is 33.8 kW/m^2 (based on experimental data), is added to obtain the total heat necessary from the molten glass, QB. QB is then placed into the heat flow matrix qz at the indices which represent the cold cap bottom, and the heat transfer from the top of the cold cap to the slurry (Htotal – QU) is added in the appropriate indices in qz0. The constants used in these calculations (evaporation heat of water, initial and final temperatures of slurry, heat flux necessary to melt dry batch to molten glass), have been obtained from the documentation that accompanied the original cold cap model (http://www.pnl.gov/main/publications/external/technical_reports/PNNL-20278.pdf, section 5.4 titled "Water Evaporation").

The next section of can7.m, lines 104 to 125, is as follows:

```
% -----------------------------------
% heat balance of each finite volume
% -----------------------------------
dTdt=zeros(N,M);

% a) inner FV (cylindrical)
k=1;
    l=1;
    dTdt(k,l)=(-AR(k)*qr(k,l)+AZ(k)*(qz0(k)+qjz0(k))-AZ(k)*(qz(k,l)+qjz(k,l)))
/ (V(k)*rho(k,l)*cpc(k,l));
    for l=2:M
        dTdt(k,l)=(-AR(k)*qr(k,l)+AZ(k)*(qz(k,l-1)+qjz(k,l-1))-
AZ(k)*(qz(k,l)+qjz(k,l))) / (V(k)*rho(k,l)*cpc(k,l));
    end
%end k=1

% b) other FV ("rings")
for k=2:N
    l=1;
    dTdt(k,l)=(AR(k-1)*qr(k-1,l)-AR(k)*qr(k,l)+AZ(k)*(qz0(k)+qjz0(k))-
AZ(k)*(qz(k,l)+qjz(k,l))) / (V(k)*rho(k,l)*cpc(k,l));
    for l=2:M
        dTdt(k,l)=(AR(k-1)*qr(k-1,l)-AR(k)*qr(k,l)+AZ(k)*(qz(k,l-1)+qjz(k,l-
1))-AZ(k)*(qz(k,l)+qjz(k,l))) / (V(k)*rho(k,l)*cpc(k,l));
    end
end
```

In this section, an NxM matrix called dTdt is created to hold the solution values of the right hand side of the heat balance ODE for each FV. The first part of this section of code finds the value of dTdt for the innermost (leftmost) FV, which is a cylinder. The second part of the code finds the value of dTdt for the FVs shaped like cylindrical rings.

The last section of can7.m is shown below.

```
% pack state variables (matrix --> vector)
dydt=zeros(N*M,1); % column vector for integration
index=0;
for k=1:N
    for l=1:M
        index=index+1;
        dydt(index)=dTdt(k,l);
    end
end
```

The above code simply turns the dTdt matrix from an NxM matrix into a column vector with N*M columns. The column vector is stored in dydt. This is necessary because the ode15s solver requires that y' values be inputted as a column vector.

Now that can7.m is finished executing, control finally returns to main2D.m. The next section of main2D.m is shown below:

```
yyy=v2array3d(yy); % simulation results yy into 3D matrix
% 1st index = axial (z)
% 2nd index = radial (r)
% 3rd index = time (t)
```

Keeping in mind that yy is a 2D solution matrix consisting of temperature values, this call to the v2array3D.m helper file is for turning yy into a 3D matrix, in which the third dimension is time. The v2array3D.m file is shown below:

```
function [yout] = v2array3D(yin)
%V2M Vector (1D) to Array (3D)
%   yin is vector
%   yout is 3d-array
M=100;
N=2;
yout = zeros(N,M,51);
dim=size(yin); % first number is the number of rows (times), the second
number is the number of columns (temperature)

for t=1:dim(1); % for all time steps

    yout(:,:,t)=zeros(N,M);
    index=0;
    for k=1:N
        for l=1:M
            index=index+1;
            yout(k,l,t)=yin(t,index);
        end
    end
end
```

The above code is just taking the 2D solution matrix and turning it into a 3D matrix. A 3D matrix in MATLAB can be thought of as layers of 2D matrices (think of it like a Rubik's cube). In this case, there is a layer for each time value, and in each layer are the ODE solutions at radial and axial coordinates.

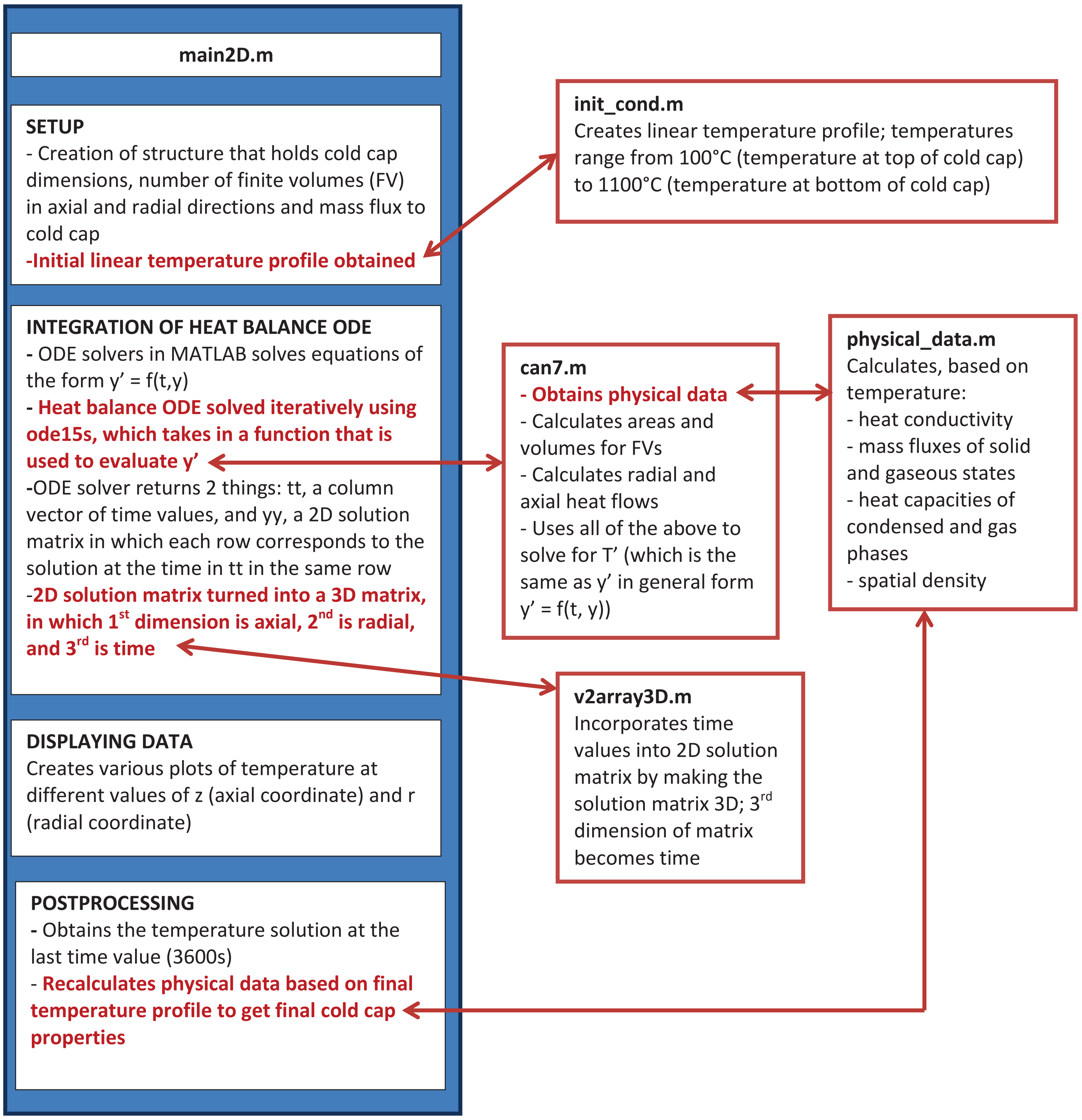
The next section of main2D.m, not shown in this document, focuses on graphing the temperature solution at different values of z (axial coordinate) and r (radial coordinate). This section can be changed to reflect whatever data is required from the model.

The last part of main2D.m, lines 76 to 87, is shown below.

```
%========================================================
%       postprocessing (results)
%========================================================
T=yyy(:,:,end); % last time
TT=T(1,1); %param.TT; % slurry temperature
% TB=T(end, end); %param.TB; % bottom temperature
physical_data % distribution of final cold cap properties
QB1 = QB;
TT1 = TT + 273.15; % Convert to K, since STAR accepts K, not C
MGAS1 = MGAS;
MGLASS1 = MGLASS;
end
```

In this code, the temperature solution at the very last time in the integration process (0.01s) is obtained and stored in the variable T. Then, physical_data.m is called again to obtain the new, final properties of the cold cap based on the final temperature values. The output variables TT1, QB1, MGAS1 and MGLASS1 are all updated so they can be sent to the STAR-CCM+ simulation in the coupled model.

Below is a flowchart outlining the interactions between the various files for the cold cap model.



main2D.m
SETUP
- Creation of structure that holds cold cap dimensions, number of finite volumes (FV) in axial and radial directions and mass flux to cold cap
-Initial linear temperature profile obtained
INTEGRATION OF HEAT BALANCE ODE
- ODE solvers in MATLAB solves equations of the form y' = f(t,y)
- Heat balance ODE solved iteratively using ode15s, which takes in a function that is used to evaluate y'
-ODE solver returns 2 things: tt, a column vector of time values, and yy, a 2D solution matrix in which each row corresponds to the solution at the time in tt in the same row
-2D solution matrix turned into a 3D matrix, in which 1st dimension is axial, 2nd is radial, and 3rd is time
DISPLAYING DATA
Creates various plots of temperature at different values of z (axial coordinate) and r (radial coordinate)
POSTPROCESSING
- Obtains the temperature solution at the last time value (3600s)
- Recalculates physical data based on final temperature profile to get final cold cap properties
init_cond.m
Creates linear temperature profile; temperatures range from 100°C (temperature at top of cold cap) to 1100°C (temperature at bottom of cold cap)
can7.m
- Obtains physical data
- Calculates areas and volumes for FVs
- Calculates radial and axial heat flows
- Uses all of the above to solve for T' (which is the same as y' in general form y' = f(t, y))
physical_data.m
Calculates, based on temperature:
- heat conductivity
- mass fluxes of solid and gaseous states
- heat capacities of condensed and gas phases
- spatial density
v2array3D.m
Incorporates time values into 2D solution matrix by making the solution matrix 3D; 3rd dimension of matrix becomes time

4. ACKNOWLEDGMENTS

The authors gratefully acknowledge that this work was supported by the Department of Energy's Waste Treatment & Immobilization Plant Federal Project Office under the direction of Dr. Albert A. Kruger. Brant Peery and Sam Alessi are also thanked for guidance on the use of ModelCenter and SharePoint websites. Eric Volpenhein of CD-adapco graciously provided STAR-CCM+ licenses for the students' use.

5. REFERENCES

[1] R. Pokorny and P.R. Hrma, "Mathematical Model of Cold Cap – Preliminary One-Dimensional Model Development," Pacific Northwest National Laboratory PNNL-20278, March 2011.