JASS v3.1 users manual

Contents

1 Introduction to JASS **2**

2 Running the simulator **3**
2.1 Basics . . . 3
2.1.1 Starting the simulator . . . 3
2.1.2 Starting and stopping the simulation . . . 4
2.1.3 Displaying simulation results . . . 4
2.2 Changing simulator properties . . . 5
2.2.1 Loading and saving plant configurations . . . 5
2.2.2 Setting the simulation speed . . . 5
2.2.3 Changing plant properties . . . 6
2.2.4 Changing model parameters . . . 7
2.2.5 The controllers . . . 7

3 Working with JassModels **12**
3.1 Introduction to JassModels . . . 12
3.2 Creating new models . . . 12
3.2.1 Modifying an existing model using JASS . . . 12
3.2.2 Using the ModelGenerator class to create new models . 12
3.2.3 Modifying an existing model using Java programming . 12
3.3 Implementing new controllers . . . 13
3.3.1 Implementing the actual controller . . . 13
3.3.2 Implementing a parameter class . . . 14
3.3.3 Implementing the controllers GUI . . . 15

4 Maintenance issues for JASS 3.1 **17**
4.1 Compiling JASS 3.1 . . . 17
4.2 Setting up a web page for running JASS 3.1 . . . 17

A Format of influent data files **19**

B Changes since version 3.0 **20**
B.1 New features . . . 20
B.2 Changes to classes in the gui . . . 20
B.3 Changes to classes not in the gui . . . 20

1 Introduction to JASS

JASS stands for Java Activated Sludge process Simulator and is a Java program that simulates activated sludge processes. Version 3.0 was developed as a Master thesis work at the Department of Systems and Control, Uppsala University. Version 3.1 was developed shortly afterwards, implementing some minor updates and new features.

In JASS the activated sludge process is simulated with the help of the International Water Association Activated Sludge Model no 1 (ASM1). The equations in the model is solved with a fourth-order Runge-Kutta method. The time step used is by default 0.01h (36s), but this value can easily be changed. Different ASP plants can be simulated with JASS. The simulation results is presented dynamically in bar diagrams and a time plot in the graphical user interface (GUI). Changes to the process can be made during simulation.

2 Running the simulator

In Figure 1 the GUI is shown. Below is a description on how the simulator is used.

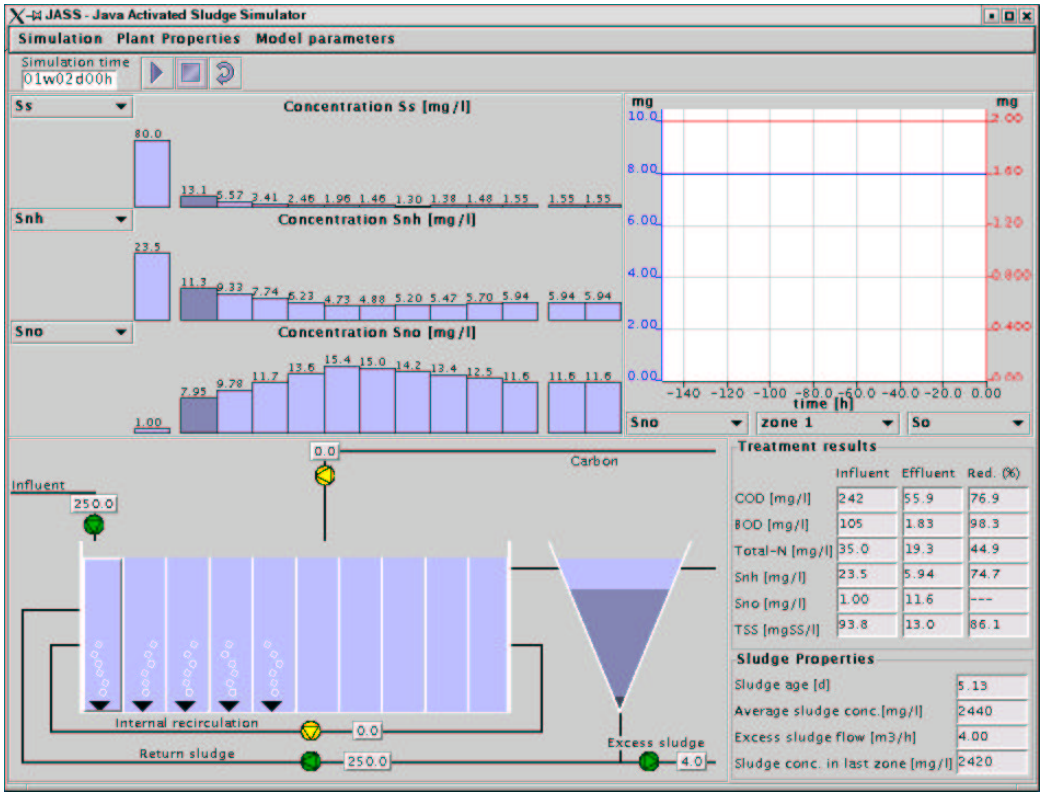

Figure 1: The Jass 3.1 GUI

2.1 Basics

2.1.1 Starting the simulator

1. The simulator can be run in any web-browser that has the Java plug-in version 1.3 or later installed. The simulator is then loaded by opening the correct URL.

2. A user that has access to the Java class files and has the Java run-time environment version 1.3 or later installed may also run the simulator as a standard Java application. The simulator is then started from the command line by invoking:

```
java -classpath <classpath> jass3.gui.JassGui
```

where `<classpath>` should be the path to the directory where the Java class files reside.

2.1.2 Starting and stopping the simulation

3. The simulated time is displayed under the label *Simulation time* in the upper left of the GUI. The time is displayed in weeks, days and hours.

4. The simulation is started by pressing the start button (the first button from the left in Figure 2) or by selecting *Start* in the *Simulation* menu. When the simulation is running, the *Start* button will be colored green.

Figure 2: The start, stop and reset buttons, from left to right.

5. Pressing the *Reset* button (the third button from the left in Figure 2) or Selecting *Reset* in the *Simulation* menu stops and resets the simulation. This means that the state of the process is now the same that it was before you started the simulation regarding the concentrations in the zones. This only affects the internal states of the compartments and the settler. Any changes made in the controllers, the influent water composition etc. are left unchanged.

6. Pressing the *Stop* button (the second button from the left in Figure 2) or selecting *Stop* in the *Simulation* menu stops the simulation temporarily. If the *start* button is pressed when the simulation is stopped the simulation will continue from where it was. The *Stop* button is colored red when the simulation is not running.

2.1.3 Displaying simulation results

7. In the upper left of the GUI are three bar diagrams. Each of these display the concentration of a component in all zones as well as in influent, effluent and recirculation water. To choose what component to display in a bar diagram, click the combo box to the left of the bar diagram, and choose a component.

8. In the upper right of the GUI, there is a plot with three combo boxes under it. This plot displays the values from the last 150 hours of two components in a specific zone. To change which components to display, use the leftmost and rightmost combo boxes. To change the zone, use the combo box in the middle. Note that the plot only stores the data that is currently being plotted, so when you change the component or zone, the plot will start again at time zero.

9. In the lower right of the GUI some treatment results and some sludge properties are displayed. The treatments results are displayed as influent and effluent concentrations together with the reduction in percent.

2.2 Changing simulator properties

10. Changes in the process are performed in a number of different dialog boxes. It is important to note that no changes take effect until the *Apply* or the *OK* button is pressed. The *OK* button also closes the dialog box. The *Cancel* button closes the dialog box without applying any changes.

2.2.1 Loading and saving plant configurations

11. The *Load* item in the *Simulation* menu can be used to load different plant configurations. When this is item selected the available plant configurations are displayed in a menu. If JASS is run in a non-applet environment, it is also possible to load a configuration from a file.

12. With the *Save* item in the *Simulation* menu the current plant configuration can be saved to a file. Note that this option is not available when JASS is run as an applet (i.e. in a web browser), since applets are not allowed to save files to disk.

2.2.2 Setting the simulation speed

13. If *Simulation settings* in the *Simulation* menu is selected, a dialog is displayed where the simulation speed and the time step used for solving the differential equations can be set. If the *Limit simulation speed* checkbox is checked the program tries to run the simulation in the speed given in the *Simulation speed* field. In this field the speed is given as simulated hours per real time hours. Note that factors outside

the program, such as the hardware, limits the maximum speed, so if the given value is to high the speed may be slower than the given value. If the *Limit simulation speed* checkbox is not checked the simulation speed is not limited by the program.

2.2.3 Changing plant properties

14. In the lower left of the GUI a schematic picture of the process is displayed. Here the compartments, the settler and the flow-paths (internal recirculation, return sludge etc) are displayed. Some flow-paths has a pump attached. If the pump is clicked, a dialog is displayed where the flow rate can be set. The color of a pump reflects the current flow rate: a green pump means the flow rate is non-zero, while a yellow pump represents a flow rate of zero. A pump may also be gray, which means that the flow rate is controlled by an automatic controller and may not be set manually. If the user clicks in a compartment or in the settler, the corresponding bar will be highlighted in the bar diagrams.

15. The composition of the influent water can be set by selecting *Influents* in the *Plant properties* menu. This displays a dialog where the concentration of the different components can be set.

16. It is also possible to read the influent component concentration from a file. This is done by checking the *Get influents from file* checkbox in the *Influents* dialog box. A pop-up menu is then displayed where some influent files are listed. If the simulator is run in a non applet environment, the *From file* item in the pop-up menu lets the user load input files not listed. An influent data file contains both the flow rate and the incoming concentrations of the components for a period of time.

17. The volumes of the compartments can be set by selecting *Compartment volumes* in the *Plant properties* menu. This displays a dialog where the volumes can be set.

18. The area, height and feed height of the settler can be set by selecting *Settler properties* in the *Plant properties* menu. This displays a dialog where the properties can be set.

19. In the *Controllers* sub-menu in the *Plant properties* menu the controllers used in the plant are listed. Selecting one brings up a dialog

where the controller parameters may be set. For more about the controllers, see Section 2.2.5.

20. In the *Flow rates* sub-menu in the *Plant properties* menu different flows in the plant are listed. Selecting one of them brings up a dialog where the flow rate can be set, just as when the corresponding pump in the process image is pressed.

21. Selecting *Miscellaneous properties* in the *Plant Properties* menu displays a dialog box where the concentration of the external carbon can be set.

2.2.4 Changing model parameters

22. Selecting *ASM1-parameters* in the *Model parameters* menu brings up a dialog where the different parameters in the ASM1 can be set. Some of the parameters are temperature dependent, and will therefore change with temperature, which can also be set.

23. Selecting *Settler model parameters* in the *Model parameters menu* brings up a dialog where the parameters for the settler model can be set.

24. Selecting *Oxygen transfer parameters* in the *Model parameters* dialog brings up a dialog where the parameters for the oxygen transfer model can be set.

2.2.5 The controllers

25. The oxygen in each compartment is controlled by a PID controller. To change the parameters for one of these, select *DO PID controllers* in the *Controllers* menu. A dialog like the one shown in Figure 3 will be displayed. The dialogs holds a tabbed panel where the number of tabs matches the number of compartment. The tabs are named *zone 1*, *zone 2* and so on. Clicking on one of the tabs displays the parameters for the DO controller of that zone. Here it is possible to change the reference value and the PID parameters. The buttons in this dialog box works a little different than in the others: after Changes has been made, pressing *Apply* will set the new parameters of the controller. If *Reset* is pressed, the values in the current tab of the dialog box will be reseted to the latest applied values. Pressing *Cancel* closes the dialog.

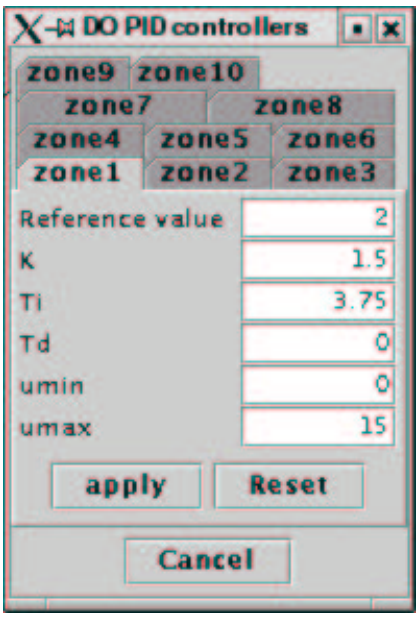

Figure 3: The DO PID controllers dialog

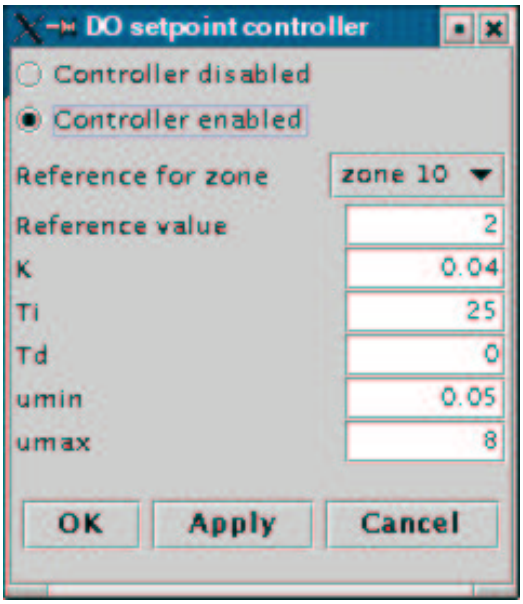

Figure 4: The DO setpoint controller dialog

26. It is also possible to use a supervisory PID for the DO setpoint. By selecting *DO setpoint controller* in the *Controllers* menu a dialog is displayed, shown in Figure 4. It displays the parameters of the DO

setpoint PID. If the *Controller enabled* radio button is selected, the supervisory controller will provide the reference values for the DO PID controllers based on the NH_4 concentration in the reference zone. The reference zone can be chosen in the dialog, as well as the reference value for NH_4 and the PID parameters.

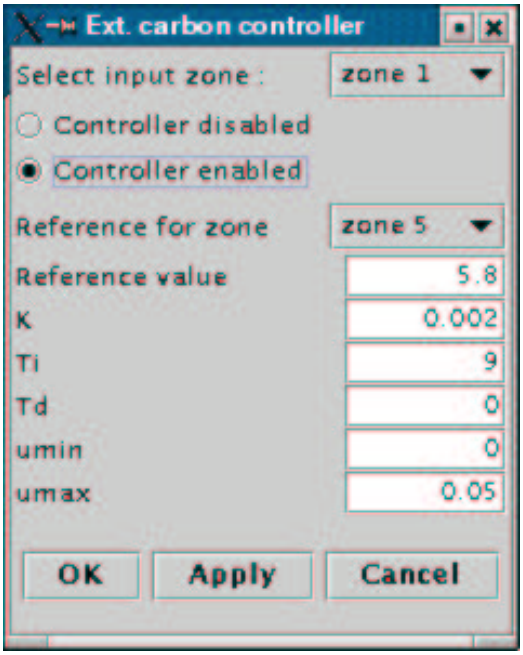

Figure 5: The Ext. carbon controller dialog

27. The carbon controller is a PID controller that controls the NO_3^- concentration in an arbitrary anoxic zone by changing the flow rate of external carbon. By selecting *Ext. carbon controller* from the *Controllers* menu the dialog shown in Figure 5 is displayed. In this dialog it is possible to set in which zone the carbon should be added, in which zone the NO_3^- concentration should be controlled, the reference NO_3^- value and the PID parameters. It is also possible to enable/disable the controller. In the latter case the carbon flow rate can be set manually in the *Carbon* dialog.

28. The internal recirculation controller controls the NO_3^- concentration in the last anoxic zone by changing the internal recirculation flow rate. The controller is a PID controller. By selecting *Internal recirculation controller* in the *Controllers* menu, the dialog shown in Figure 6 is displayed. Here it is possible to set the parameters of the controller.

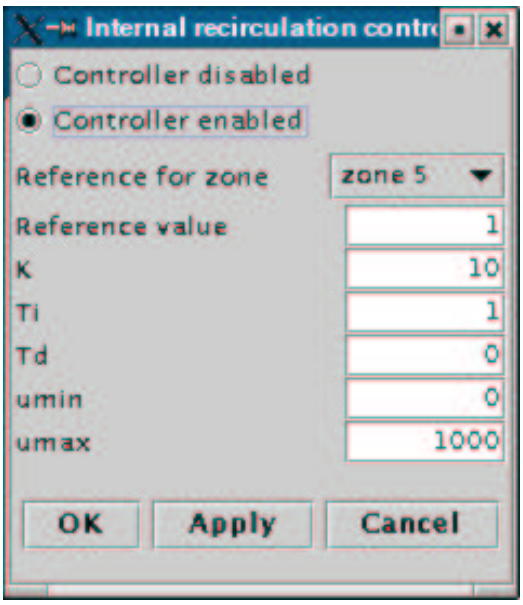

Figure 6: The Internal recirculation controller dialog

The controller may also be disabled to allow for manual setting of the internal recirculation flow rate.

29. The excess sludge controller keeps a fixed sludge age by controlling the excess sludge flow rate. It uses a mass balance equation to calculate the flow rate. By selecting *Excess sludge controller* in the *Controllers* menu the dialog shown in Figure 7 is displayed. In this dialog settings for the excess sludge controller is displayed. The controller can be set to manual mode, which allows for setting the excess sludge flow rate by clicking on the corresponding pump symbol. If the flow is set to automatic control, the desired sludge age should be given.

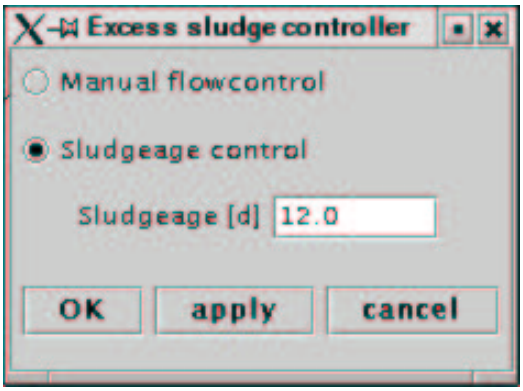

Figure 7: The Excess sludge controller dialog

3 Working with JassModels

3.1 Introduction to JassModels

A JassModel is a model representing the activated sludge process setup, and is encapsulated in the Java class JassModel.

3.2 Creating new models

3.2.1 Modifying an existing model using JASS

One method for creating a new model is to load an old model into JASS, modifying it and then saving it. This method has a limited use, since there are many parts of the process setup that can not be changed. These include the number of compartments, the types of controllers and more. What can be changed are the compartment volumes, the ASM1 parameters, the settling parameters and parameters for the controllers, and also the composition of the influent water. Also all flows like influent, internal recirculation etc can be changed.

3.2.2 Using the ModelGenerator class to create new models

The Java class `jass3.simulation.ModelGenerator` can be used to create new models. Invoking it from the command prompt by typing

```
java -classpath <classpath> jass3.simulation.ModelGenerator -h
```

where `<classpath>` is the java classpath displays the possible command line parameters. The number of compartments and their volume can be set on the command line, and the plant can be post- or predenitrifying. A model created this way can then be modified further in the simulator.

3.2.3 Modifying an existing model using Java programming

A user that has some experience of Java programming can use Java programming to modify an existing model. The existing model can be a serialized (saved) model or an Instance of class JassModel or one of its subclasses. The advantage of this method compared to the one described above is that it gives the user the possibility to add and remove controllers and flowgenerators, even new types of controllers and flowgenerators can be added to the model (see Section 3.3 for more info on this). If you start with a saved model, it is not possible to change the number of compartments. To create a model with any number of compartments, you can either instanciate

JassModel directly to create an empty model with the given number of compartments. The model is empty in the sense that it has no FlowGenerators and no controller, so it will be necessary to add those that are needed. A simpler way is to instanciate the class OldJassModel which creates a model with a given number of compartments, and also adds standard features such as return sludge flow, internal recirculation, aeration and such.

To modify a model in this way, the user writes a program that creates an instance of JassModel or one of its subclasses, either by instantiating an existing class or by loading a saved model. Then the desired changes to the model can be done, and then the model should be saved. Methods for saving and loading models are implemented in the java class ModelFileHandler. It is of course also possible to copy the source code from a class that inherits JassModel, and then modify and rename it to get a new type of model.

3.3 Implementing new controllers

Implementing a new type of controller should be fairly easy. Normally it is necessary to implement three classes:

1. A class representing the actual controller
2. A class representing the controllers GUI
3. A parameter class used for passing data between the controller and the GUI

The two last items can be omitted, but only if no properties of the controller should be changed during simulation.

3.3.1 Implementing the actual controller

The class implementing the controller should be a subclass of `Controller`. The most simple case is implementing a single input single output controller. In this case, it should be a subclass of `SISOController`. To make the controller functional, the abstract method `newInput( double input )` should be overridden. This method takes an input, most likely a sampled value of the variable to be controlled, and should then calculate a new output signal that should be stored in the double field `u`. The out signal can then be accessed by calling the `getValue()` method. For example, say that one wishes to implement a P controller. The class could be called PController and the Java code would look like this:

```
import jass3.simulation.controller.*;
public class PController extends SISOController
                  implements java.io.Serializable{
   /* k is the K parameter of the P controler */
   private double k;

   public PController(){
      k = 2;
   }

   public void newInput( double input ){
      double e = yref - input;
      u = k * e;
   }

}
```

where `yref` is a double field defined in `SingleOutController` that holds the reference value. We now have a functional controller: when `newInput` is called it calculates the control error and then a new out signal, that we can read with `getValue()`. The `implements java.io.Serializable` part makes it possible to save the controller to a file. This is necessary to be able to save a JassModel using the controller.

Also if the controller has a GUI, the methods `hasGui()` and `getGui()` from class `Controller` should be overridden. The first one should return true, and the other on should return an instance of the GUI class.

3.3.2 Implementing a parameter class

JASS 3.0 uses special parameter classes for passing data between a controller and its GUI. Such a class is needed for every controller that has a GUI. The parameter class should be a subclass of `Controller.Parameters`. Note that this is an inner class of `Controller`. Making the parameter class an inner class is a good idea. The parameter class should hold data representing the properties of the controller. For example, if a parameter class were written for `PController` it would hold information about the K value and the reference value. If it were implemented as an inner class the code for `PController` would look like:

```
import jass3.simulation.controller.*;
public class PController extends SISOController
              implements java.io.Serializable{
```

```
   /* k is the K parameter of the P controler */
   private double k;

   public static class Parameters extend SingleOutController.Parameters{
      public double k;

      public Parameters( double ref, double k ){
         super( ref );
              this.k = k;
      }
   }

   public PController(){
      k = 2;
   }

   public void newInput( double input ){
      double e = yref - input;
      u = k * e;
   }
}
```

The parameter class here inherits `SingleOutController.Parameters`. It could also have inherited `Controller.Parameters`, but then a field holding the reference value would have to be added, since the `ref` field would not have been inherited.

3.3.3 Implementing the controllers GUI

The GUI should be a subclass of `ControllerGui`. This class has two abstract methods that will have to be implemented: `getData()` and `setParameters( Controller.Parameters parameters )`. `getData()` should return the input data from the GUI, encapsulated in the parameter class of the controller. `setParameters` should set the GUI element to reflect the current values of the controllers properties. For example, a GUI to the class PController could be a class called `PControllerGui`. It could have two `jass3.gui.components.DecimalField` for the reference value and the K value respectively, called `refField` and `kField`. The implementation of the above mentioned methods would then be:

```
public Controller.Parameters getData(){
   return new PController.Parameters( refField.getValue(),
                                      kField.getValue() );
}

public void setParameters( Controller.Parameters parameters ){
   PController.Parameters p = (PController.Parameters) parameters;
   refField.setValue( p.ref );
   kField.setValue( p.k );
}
```

4 Maintenance issues for JASS 3.1

4.1 Compiling JASS 3.1

A Makefile is provided for JASS 3.1 that makes it easy to compile. A Java compiler of at least version 1.3.1 should be used. Before compiling, some parameters need to be set in the makefile:

- JAVAC should be set to the path to the Java compiler.
- JAVA should be set to the path to the Java executable.
- SOURCEPATH should be set to the path to the source code.
- DESTPATH should be set to the path where the class files should be put.
- (optional) DOCSPATH should be set to the path where the javadocs should be put if they are generated.

When this is done, simply typing `make` or `make all` in the directory where the makefile resides compiles all classes. It is also possible to compile only one class by typing `make classname.class`. This will also compile dependents of the class if their source file has changed. Some other useful make-ables are also present:

- `make resources` should be called when the resource files have been changed. This will copy the resource files in the source directory to the class directory.
- `make clean` removes all class files.
- `make modeljar` will create a jar file containing some common models.
- `make docs` will create the javadoc documentation of the classes and put them in the directory pointed to by DOCSPATH.

There are some other make-ables as well, but they are probably less useful.

4.2 Setting up a web page for running JASS 3.1

To create a web page containing the applet JASS 3.1 it is necessary to first create a standard applet html page, and then run the *HtmlConverter* that comes with the sun jdk on the html file. In the same directory as the html file, there should be a jar-file named *models.jar* containing one or more saved

models. Also, to make it possible to load influent data from file, there should be a jar-file named *influent_data.jar* containing one or more text files with influent data. For information of the format of the influent data files, see Appendix A

JASS 3.1 takes three applet parameters, these are listed and described in Table 1. If resources for another language than English and Swedish should be implemented, the country and language parameters could be set to appropriate values.

country	Should be set to US for English version, and SE for Swedish version.
language	Should be set to en for English version, and sv for Swedish version.
defaultmodel	The name of the plant setup to load when the program is started. The name should be the name of one of the models contained in the models.jar file.

Table 1: JASS 3.1 applet parameters

A Format of influent data files

The data in an influent data file should be separated in columns, separated with spaces or tabs. The first row should contain header for each column, indicating what data is in the column. The valid column headers are listed in Table 2, note that the case is ignored. Note that if one type of data is not present, its value will be zero all the time. Therefore flow rate has to be present, unless it should be zero all the time. A column for time also has to be present and the row must be sorted in time order. Also note that since the X_P ASM1 state variable is not present in JASS, the values for X_P and X_I are added together and used as value for X_I.

header	meaning
xbh	The ASM1 $X_{B,H}$ component $[g \cdot m^{-3}]$
xba	The ASM1 $X_{B,A}$ component $[g \cdot m^{-3}]$
xi	The ASM1 X_I component $[g \cdot m^{-3}]$
xs	The ASM1 X_S component $[g \cdot m^{-3}]$
xp	The ASM1 X_P component $[g \cdot m^{-3}]$
xnd	The ASM1 X_{ND} component $[g \cdot m^{-3}]$
si	The ASM1 S_I component $[g \cdot m^{-3}]$
ss	The ASM1 S_S component $[g \cdot m^{-3}]$
so	The ASM1 S_O component $[g \cdot m^{-3}]$
sno	The ASM1 S_{NO} component $[g \cdot m^{-3}]$
snh	The ASM1 S_{NH} component $[g \cdot m^{-3}]$
snd	The ASM1 S_{ND} component $[g \cdot m^{-3}]$
salk	The ASM1 S_{ALK} component [Molar units]
q	The Flow rate $[m^3 \cdot d^{-1}]$
t	The time [d]

Table 2: The valid column headers in influent data files.

B Changes since version 3.0

B.1 New features

- More simulation parameters can now be changed. The new ones include the simulation speed, the integration time step, the settlers feed layer, the settler model threshold concentration, parameters for the oxygen transfer function and the concentration of external carbon.
- Influent data regarding concentrations and flow rates may now be read from file.

B.2 Changes to classes in the gui

- The classesjass3.gui.GuiMain and jass3.Jass3 has both been replaced by jass3.gui.JassGui, which is now the main class.
- The class jass3.gui.dialogs.OxygenTransferParametersDialog has been added and is used in the gui to allow setting of oxygen transfer parameters.
- The class jass3.gui.dialogs.SettlerPropertiesDialog has been added and is used in the gui to allow for setting of settler properties.
- The class jass3.gui.dialogs.SettlerModelParametersDialog has been added and is used in the gui to allow for setting settler model parameters.
- The class jass3.gui.dialogs.ASM1ParametersDialog replaces the old ProcessParametersDialog.
- Class jass3.gui.dialogs.SimulationSettings dialog has been added and is used in the gui.
- Minor cosmetic changes has also been made.

B.3 Changes to classes not in the gui

- The class jass3.simulation.iawqasm.State has been renamed to jass3.simulation.iawqasm.ASM1State.
- The class jass3.simulation.iawqasm.Flow has been completely removed.

- The class jass3.simulation.iawqasm.ProcessParameters has been removed and is replaced by jass3.simulation.iawqasm.ASM1Parameters and new inner class IAWQSettler.ModelParameters.
- Minor changes has been made to jass3.simulation.flowgenerator.FlowGenerator and its subclasses due to the removal of the class jass3.simulation.iawqasm.Flow.
- New class jass3.simulation.flowgenerator.AirFlowGenerator has been added.
- Minor changes has been made to jass3.simulation.controller.Controller and its subclasses to make it easier to allow for turning controllers on and off.
- Inner class jass3.simulation.IAWQCompartment.OxygenTransferParameters has been added.
- Inner class jass3.simulation.IAWQSettler.Properties has been added.
- Inner class Simulator.Settings has been added.