

## MATSim Tutorial

# Getting started with MATSim

Release 2009-01-08 (r4776)

## Content

Introduction

Description of a Scenario

Running the Examples

*The “equil” Scenario, Running a Single Iteration, Visualizing the Simulation-Results, Reading Events, Modifying the Settings, Running Multiple Iterations, Modifying the Re-Planning, Using an External Re-Planning Module, Using an External Mobility Simulation*

Creating a Custom Controller

*Reading network and population, Setting up Events, Running the Mobility Simulation, Writing Visualizer Output, Using the Configuration Settings, Scoring Plans, Running Multiple Iterations, Adding Re-Planning*

What's next?

## Introduction

MATSim provides a toolbox to implement large-scale agent-based transport simulations. The toolbox consists of several modules that can be combined or used stand-alone. Modules can be replaced by own implementations to test single aspects of your own work. Currently, MATSim offers a toolbox for demand modeling, mobility-simulation, re-planning, a controller to iteratively run simulations as well as methods to analyze the output generated by the modules.

This tutorial will make you familiar with MATSim. You will learn

- how to run and simulate the provided sample scenario,
- to understand the configuration settings so you can change them correspondingly for your own scenarios,
- how to integrate an external, custom mobility simulation,
- how to integrate a custom, external re-planning module.

Finally, we will take a look at the code so you will be able to extend the previously mentioned examples to create further sophisticated simulations.

MATSim is written in Java 1.5. You will need at least the Java Runtime Environment (JRE) 5 or newer to run the examples and the Java Development Kit (JDK) 5 or newer to compile your own code that uses MATSim functionality.

## Description of a Scenario

A scenario always consists of at least a network and a population with plans. The network is a representation of a physical road network, on which traffic happens. The population is a collection of agents, in which each agent has a plan that describes activities and routes through the network.

Additionally, a scenario can include additional input files (e.g. a world or a description of facilities) that are referenced by other data. A configuration file collects all settings including file paths to the data files (network, population, ...). The settings are separated into *modules*, which group related settings together.

## Running the Examples

### The “equil” Scenario

The following examples all use the very simple *equil* scenario, consisting of the two files *network.xml* and *plans100.xml*. It is located in *examples/equil*. Looking at the network in the included visualizer may be the easiest way to get an impression of the network. Additionally, you can look at the textual description of the network in the XML file. Use the following command to start the visualizer:

```
java -cp MATSim_r4776.jar org.matsim.utils.vis.netvis.NetVis
```

After the visualizer started, you will be prompted to choose a file. Set the File Format to “All Files” and select the file *network.xml*. You can zoom in the visualizer either by pressing the “+” button in the toolbar or by spawning a rectangle when dragging the mouse pointer across the screen with the mouse button pressed. To zoom out, use the “-” button in the toolbar. You can scroll the image either by using the scroll bars (on the left and on the bottom side of the window) or by holding the middle mouse button and moving the mouse. Also check out the other settings available in the toolbar.

### Running a Single Iteration

In this example, 100 agents will start on link 1 and first travel to link 20 by crossing the nodes 2, 7 and 12. A bit later, the agents will depart at link 20 and travel back to link 1 by crossing the nodes 13, 14, 15 and 1. Have a look at *plans100.xml* to get an impression on how the agents and their plans are described.

To run the simulation, use the following command:

```
java -cp MATSim_r4776.jar org.matsim.run.Controller configs/singleIteration.xml
```

If everything works as expected, you should see several lines of status output. If you look through this log, you will find messages about network and plans being read, the iteration being run, scoring information and shutdown messages. Please ensure at the end of the log that the message “shutdown completed” does not include the word “unexpected”. If it reads “unexpected shutdown completed”, an error or exception has happened. In this case, you will find more information in earlier lines of the log.

### Visualizing the Simulation-Results

Start again the Visualizer, but this time select the file *output/ITERS/it.0/SnapshotCONFIG.vis*. You should now again see the network, but the toolbar has additional buttons that let you control the displayed time of day. Change the time to 06:00 and increase the linewidth, until the links are no longer just thin lines, but actually are white rectangles with black borders. If you have checked “Agents” in the toolbar, you should now also see some blue dots representing the single agents.

Click “Play” to watch the agents move through the network, or increment the time in single steps using the buttons in the toolbar.

## Reading Events

Events are very simple information units which are generated by the mobility simulation. Open the events-file `output/ITERS/it.0/0.events.txt` in a text editor and look at the possible events. You may want to filter out the events of one single agent to get a better understanding of the events. You can use the events-file for further, custom analysis of the simulation.

## Modifying the Settings

Open the configuration file `configs/singleIteration.xml` in a text editor and look at the various settings (You may want to ignore the settings in module *planCalcScore* for now). Try to change some settings in the module *simulation* (e.g. setting an `endTime` of 07:00 or changing the `snapshotperiod`) and run the simulation again. You may also want to visualize the output again to see if the simulation really did what you intended to do it.

Note: The simulation will not overwrite any files in the specified output-directory. It will not even start if the output-directory is not empty. Make sure that the output-directory is either completely empty or is completely missing (though the direct parent of the specified output-directory must exist). If the directory is missing, the simulation will create the directory for you.

If you have Google Earth™ installed on your machine, you may want to change the `snapshotFormat` to “googleearth”. If you re-run the simulation, you will find the file `0.googleearth.kmz` in `output/ITERS/it.0/`, which can be opened in Google Earth.

## Running Multiple Iterations

Run the following command:

```
java -cp MATSim_r4776.jar org.matsim.run.Controler \
    configs/multipleIterations.xml
```

This will run 10 full iterations. In each iteration, 10% of the agents will try to find a faster path to travel from link 1 to link 20. Look at some of the iterations in the visualizer and try to understand what's going on. Then look at the configuration file and compare it to the `singleIteration.xml`-file.

Change the number of iterations and run it again. If the number of iterations was chosen high enough, all alternative routes should have about the same amount of traffic / congestion at peak time.

## Modifying the Re-Planning

Change the *ModuleProbability\_2* in the `multipleIterations.xml`-file to a higher value, e.g. 0.9, and change *ModuleProbability\_1* to a lower value, e.g. 0.1. Run the simulation again and look at some consecutive iterations in the visualizer. What effect can you observe?

You can also change the strategy being used: Replace the value of *Module\_2* with *TimeAllocationMutator*. Run the simulation again and have a look at the results. What does the *TimeAllocationMutator* do?

You can also combine the *ReRoute*- and the *TimeAllocationMutator*-strategy: Create two additional parameters in the module *strategy* with the names *ModuleProbability\_3* and *Module\_3* and assign this strategy the *TimeAllocationMutator*, while *Module\_2* remains with *ReRoute*. Run the simulation again and have a look at the results.

With such a setup (*BestScore*, *ReRoute*, *TimeAllocationMutator*), you could also simulate larger scenarios.

## Using an External Re-Planning Module

What if you want additional re-planning strategies, e.g. location choice? You can implement such modules on your own and use them together with MATSim. You have two possibilities for creating your own re-planning module:

- Write the module in Java and make use of functionality provided by MATSim. In this case, you're advised to integrate your module at code-level. This means you should be able to directly access and modify code of MATSim.
- Write the module in Java or any other language independently of MATSim. Your module should then run stand-alone. This way, you could also just write a wrapper to use existing code.

We will only look at the second possibility here. If you are interested in creating your module directly in MATSim, please have a look at the chapter "Creating a Custom Controller" to get an idea how and where you can include your own module.

Your external re-planning module must be an executable. MATSim will call your executable with the path of a configuration file as its only argument. Your executable thus must be able to parse MATSim configuration files in order to be used with MATSim. In the configuration file, the path to a plans file is set (module *scenario*, parameter *inputPlansFilename*) that contains the plans that your module should read and modify. A second parameter *workingPlansFilename* contains the path to a file where you must write the modified plans to. Additionally, *workingEventsTxtFilename* and *networkFilename* contains the paths to the events and network file which may be of use to your module.

To instruct MATSim to use your module for re-planning, you can add an entry to the configuration file. Have a look at the module *strategy* in `configs/externalReplanning.xml` to see how MATSim can use external modules.

We provide for testing reasons a very simple external re-planning executable. The executable is a simple jar-file containing a single Java class. The example code just takes the given plans as input and writes them out unmodified to the specified output location. The source code for the example is available within the jar-file.

Run the provided example with the following command:

```
java -cp MATSim_r4776.jar org.matsim.run.Controller \
    configs/externalReplanning.xml
```

In the log, you should see entries that the external executable was called. Additionally, any output of the external executable is written to its own log file in the corresponding iteration-directory (`output/ITERS/it.*/*`).

## Using an External Mobility Simulation

What if you already have your own mobility simulation, but are missing the agent optimization / re-planning part? Well, you can use your own mobility simulation together with MATSim as long as you are able to read in our plans and network and generate events.

The external mobility simulation will be called with a configuration file as its only argument. In the configuration file, you find parameters describing the location of the network file, the location of the plans file, and the location of the events file. Your mobility simulation should read network and plans and generate events.

We provide for testing reasons a very simple example of an external mobility simulation that just outputs a few hard-coded events. You can run the example with:

```
java -cp MATSim_r4776.jar org.matsim.run.Controller configs/externalMobsim.xml
```

Have a look at the configuration file to understand how the run is configured to use the external mobility simulation.

If you intend to write your own mobility simulation, please read the documentation about events in the Developer's Guide (<http://www.matsim.org/docs/devguide>). The events are a very important aspect in MATSim as they provide feedback from the mobility simulation to the re-planning parts of MATSim. Without (or with wrong) feedback from the mobility simulation, no useful re-planning is possible.

## Creating a Custom Controller

For simple scenarios, the configuration file offers enough flexibility to specify different simulation set-ups. But as soon as you want to run more complex scenarios or want to better integrate your own code into MATSim, you will have to get known the important classes of MATSim so you can use them in your code, and better inject your code into MATSim. In the following sections, you will learn how to implement your own controller, which you will be able to extend as you like it. While creating your own controller, you will learn enough about MATSim classes and the “philosophy” behind MATSim that you should be able to understand most of our code.

### Reading network and plans

In a first step, we will just load some the plans and network and run them once through the mobility simulation. Start by creating a new class `MyController` that contains a static main method.

At the moment, we will hard code required filenames into the code so it is easier to understand now, but switch later to using configuration files. Define some final String variables for both the network and the plans filename:

```
final String netFilename = "./equil/network.xml";
final String plansFilename = "./equil/plans100.xml";
```

MATSim has a special class `Gbl` that acts as a repository for global objects. The configuration settings are such a global object, but also the “world”. The world maintains references to important objects like the network that are used in the current scenario being run. We will first create the world, so we can later register the network with the world, and also create a default configuration object:

```
Config config = Gbl.createConfig(null);
World world = Gbl.createWorld();
```

MATSim provides both data structures and corresponding readers and writers. We will first create the data structure to hold the network and then read it with a specialized reader:

```
NetworkLayer network = new NetworkLayer();
new MatsimNetworkReader(network).readFile(netFilename);
world.setNetworkLayer(network);
```

For the plans, the code looks similar:

```
Population population = new Population();
new MatsimPopulationReader(population).readFile(plansFilename);
```

### Setting up Events

The mobility simulation will create events, for which we have to provide a data structure as well.

```
Events events = new Events();
```

Events cannot be stored within this data structure because of the huge amount of events a large-scale simulation is likely to generate. Instead, the events are processed on-the-fly by so-called events handlers. We will add such a handler that just writes all the events to a file:

```
EventWriterTXT eventWriter = new EventWriterTXT("./output/events.txt");
events.addHandler(eventWriter);
```

## Running the Mobility Simulation

Finally, we can instantiate a mobility simulation and run it. In this example, we will use the QueueSimulation:

```
QueueSimulation sim = new QueueSimulation(network, population, events);
sim.run();
```

After the simulation was run, we have to close the event writer to ensure all data is flushed to disk and that the file is properly closed:

```
eventWriter.closeFile();
```

You should now be able to compile and run this example. You can also find the complete code for this example in `src/MyController1.java`. You can compile the code with:

```
javac -cp MATSim_r4763.jar:src src/MyController1.java (On Windows, replace : with ;)
```

and can run it with:

```
java -cp MATSim_r4763.jar:src MyController1 (On Windows, replace : with ;)
```

Make sure the directory `output` exists; otherwise the events cannot be written.

## Writing Visualizer Output

If you run the example above, all you can see is some information output on the console and the written events. To make it more interesting, we add now output for the visualizer. Before running the mobility simulation (`sim.run()`), open a writer for the visualizer:

```
sim.openNetStateWriter("./output/simout", netFilename, 10);
```

This command will advise the QueueSimulation to write the network state every 10 seconds to a file beginning with "simout".

We can start the visualizer automatically after the simulation finished with:

```
String[] visargs = {"./output/simout"};
NetVis.main(visargs);
```

When you run this example (you can find the complete code for it in `src/MyController2.java`), the visualizer should open automatically after the simulation finished.

## Using the Configuration Settings

We can set the start and end time of the simulation by adjusting the configuration settings:

```
config.simulation().setStartTime(Time.parseTime("05:55:00"));
config.simulation().setEndTime(Time.parseTime("08:00:00"));
```

Add these lines before the QueueSimulation is instantiated to have an effect on the simulation.

Instead of setting all configuration settings manually in the code, we can also use a configuration file. To read a configuration file, replace the call to `Gbl.createConfig(null)` with:

```
Gbl.createConfig(new String[] { "./configs/myConfig.xml" });
```

Now you can change settings like the start or end time of the simulation in the file `configs/myConfig.xml`. You find the example code in `src/MyController3.java`.

## Scoring Plans

For each plan, a score can be calculated that reflects how well the plan performed during the simulation. If the agent traveled a long time, maybe was even stuck in a traffic jam, the score will be slower than when the agent could travel with free flow speed and had more time to perform activities.

Add the following lines to your code before the simulation is run:

```
CharyparNagelScoringFunctionFactory factory = new
    CharyparNagelScoringFunctionFactory();
EventsToScore scoring = new EventsToScore(population, factory);
events.addHandler(scoring);
```

and the following line after the simulation is run:

```
scoring.finish();
```

EventsToScore collects events and uses them to calculate how long the agent was traveling and how long the agent was performing activities. The scoring function calculates the actual scores for traveling or performing an activity. We provide a default scoring function, CharyparNagelScoringFunction, to calculate the effective scores. This two-tier approach makes it possible to only replace the calculation of the score with a more specialized function without having to re-implement all the event-handling functionality. Because the scoring object cannot know when the last event for an agent is handled and thus when the final score for an agent can be calculated, a call to `scoring.finish()` is necessary. In this call, the final scores for all agents are calculated and assigned to the executed plans.

To calculate the scores, the scorer needs additional settings, which can be done in the configuration file. Make sure that your code loads the configuration file `configs/myConfigScoring.xml`.

You can calculate the average score of all plans with:

```
PlanAverageScore average = new PlanAverageScore();
average.run(population);
System.out.println("The average score is " + average.getAverage());
```

You find the complete example in `src/MyControler4.java`.

## Running Multiple Iterations

If you want to run multiple iterations, you will need to encapsulate the call to the QueueSimulation in a loop:

```
for (int iteration = 0; iteration <= 10; iteration++) {
    QueueSimulation sim = new QueueSimulation(network, population, events);
    sim.openNetStateWriter("./output/simout", netFilename, 10);
    sim.run();
}
```

But this is not yet very interesting, as in every iteration the exact same plans are executed. We need to define some re-planning that modifies the plans between runs of the mobility simulation.

## Adding Re-Planning

Re-planning is a central concept in MATSim, as it is in all agent-based simulation. Without re-planning (sometimes also called agent-learning) the agents would perform the same plans in every iteration. Re-planning is done by applying so-called strategies to the plans. The javadoc for the package `org.matsim.demandmodeling.replanning` is a good starting point to get accustomed to the terms used to describe re-planning. A strategy can consist of zero or more strategy modules that modify the plans. A strategy manager coordinates the re-planning process. Start by creating a new StrategyManager and adding two strategies to it:

```
StrategyManager strategyManager = new StrategyManager();
PlanStrategy strategy1 = new PlanStrategy(new BestPlanSelector());
PlanStrategy strategy2 = new PlanStrategy(new RandomPlanSelector());
strategyManager.addStrategy(strategy1, 0.9);
strategyManager.addStrategy(strategy2, 0.1);
```

As an agent can have more than one plan, we have to tell the strategy which plan should be modified. The first strategy will select the plan with the best score from the agent, while the second strategy will

select a random plan from the agent. When adding the strategy to the strategyManager, a weight has to be specified that defines how likely an agent will be modified with the added strategy.

Now we can add a strategy module to one of the strategies:

```
TravelTimeCalculatorArray ttimeCalc =
    new TravelTimeCalculator(network);
TravelTimeDistanceCostCalculator costCalc =
    new TravelTimeDistanceCostCalculator(ttimeCalc);
strategy2.addStrategyModule(
    new ReRouteDijkstra(network, costCalc, ttimeCalc));
events.addHandler(ttimeCalc);
```

The actual strategy module added is the ReRoute-module. This module calculates the fastest route through the network from one location to another. Because the travel time on the network can vary depending on how many other agents are on the road (maybe causing traffic jams), the ReRoute-module needs some way to figure out the actual travel time on a link at a specified time. That's what the TravelTimeCalculator is for: It analyzes the events from the mobility simulation and calculates the actual travel time on the links. This is why it has to be added to the events as a handler. The ReRoute-module will then query the TravelTimeCalculator when it has to find the fastest route. Additionally, the routing process needs link costs to find the cheapest route. The TravelTimeDistanceCostCalculator calculates link costs based on the actual travel times and the distance.

The first strategy remains without an actual StrategyModule. It will just select the plan with the best score of an agent for the next execution of the mobility simulation.

In the iteration loop, we can now call the strategyManager to re-plan the agents after the mobility simulation ran and the score is updated:

```
strategyManager.run(population);
```

If you run this example (you can find it in `src/MyController5.java`), you should see a similar output than in one of the previous, config-file only, examples.

## What's next?

Congratulations, you have just written your first useful MATSim controller. You can now start to extend it, add your own re-planning modules or implement your own mobility simulation. A good starting point may be to improve your current controller by eliminating some of the flaws. As an example, currently the events of each iteration overwrite the previous iteration's events. You may want to create a new eventWriter for every iteration. In that case, do not forget to remove the old eventWriter with `events.removeHandler(eventWriter)`. Or you may just re-init the eventWriter with a new filename: `eventWriter.init(newFilename)`.

Another possibility is to move some hard coded settings to the configuration file and using `config.getParam(moduleName, paramName)` to access the values. The number of iterations may be a good example of a value that could be specified in the configuration file.

You can find additional documentation on our website in the developer's guide at [www.matsim.org](http://www.matsim.org).

If you plan to set up your own scenario, you need a network and a description of a population. Have a look at the provided example files to get a better understanding of the file formats. The creation of the population can be based on a variety of data (census, commuter matrices, ...), whatever is available. The page "Building a new Scenario" (<http://matsim.org/docs/new-scenario>) gives further information about that topic.

Have fun with MATSim! We would be happy to hear from you what for you are using MATSim.