# MATSim How To's

V 06_09_14

## Introduction

MATSim is a framework for the simulation of traffic. Parts of it can be used as a framework for demand modeling. Or one can just use parts of it, e.g. XML parsing, as utility functions.

The MATSim package is divided into two relevant parts:
1.) agent database (fed by initial plans from the Demand modeling framework)
2.) physical simulation.

These two parts rely on each other in the following notion:
The agent database is responsible for generating/loading plans, which are then "sent" into the physical simulation. The simulation (mobsim) will generate "events", which will in turn be "fed back" into the agent database for scoring/changing the plans. This process is iterated until some end condition is reached (e.g. a given number of iterations).

In the following chapters, we will introduce you to how to use MATSim as a tool for iteration over plans and how to code your own additional functionality in the MATSim framework.
Right now the source code for MATSim is not released yet. So only the last chapters concerning the use of MATSim with config files can actually be tried out. But you should read the other chapters as well for getting started and getting known how MATSim works internally.

## Using MATSim without config files

In the first example, we will load some plans and run them through the mobility simulation.
We will need two files:

1.) A plans file. This is an XML file that describes the plans of our "population".
2.) A network file, describing the underlying traffic network.

In our example we use the files "equil_plans.xml" and "equil_net.xml".

We create a class `StandaloneSimRun` to run the simulation. It only contains a static main method.

In the class , we have to define a "global" world instance in which we can register our plans, network, etc.

```
World world = World.createSingleton();
```

We will have to load both, network and plans, through an appropriate loader class and register them with the world instance:

```
QueueNetworkLayer network = new QueueNetworkLayer();
// Read network file with special Reader Implemetation
```

```
NetworkReaderI netReader = new NetworkParser(new
          NetworkReaderHandlerImplV1(network));
netReader.readfile(netFileName);
world.setNetworkLayer(network);

Plans population = new Plans();
// Read plans file with special Reader Implemetation
PlansReaderI plansReader = new PlansParser(new PlansReaderHandlerImplV4(population));
plansReader.readfile(popFileName);
world.setPopulation(population);
```

Then we have to create an event class instance (which can be a dummy implementation in this case) and an instance of the chosen simulator, in this case the QueueSimluation.

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

We add a simple writer to the events, which will receive every generated event an just dump it to a plain text file.

Finally, we run the simulation with:

```
sim.run(net, pop, events);
```

When we run the example, all we can see is some informational output from the simulation run.
So we should try something more interesting: We should write the status of the simulation to a file that can used for visualization.
For this we will tell the QueueSimulation to construct a visualization file writer and after having run the simulation we will visualize the outcome.
Use the method call `QueueSimulation.openNetStateWriter(String snapshotFilename, String networkFilename, int snapshotPeriod)` to prepare for writing the file, e.g.:

```
sim.openNetStateWriter("simout", netFileName, 10)
```

This command will write the network-state every 10 seconds to a file beginning with "simout4Viz" (the letters "4Viz" are appended by the QueueSimulation automatically).

As a last command after running the simulation, we will create a viewer for the output and tell it where to find the file.

```
String[] visargs = {"simout4Viz"};
NetVis.main(visargs);
```

Now we should see the network being visualized in the NetVis Viewer. As the first agents will not move before 05:55:00, nothing can be seen in the time before. We can jump in time to 05:55:00 by editing the counter in the toolbar above the net graph.

We will use another bit of our toolkit to change start and end time of our simulation run. By default every simulation run starts at midnight (00:00:00) and runs until 23:59:59. We can override the times in our configuration system to shorten the simulation times.
With the method `Config.getSingleton()` we can get the Configuration-Singleton, which handles any setting we want to pre-define. This configuration is normally parsed in from an XML-

type config file, as we will see in a later tutorial, but we can also set parameters by hand.

With

```
Config.getSingleton().setParam(Simulation.SIMULATION, Simulation.STARTTIME,
          "05:55:00");
Config.getSingleton().setParam(Simulation.SIMULATION, Simulation.ENDTIME,
          "08:00:00");
```

we can set start and end time by hand. The QueueSimulation will use these times.

## Replanning

In this section we will see how to make several iterations and to add some replanning modules to the execution.

What are iterations?
Right now we use the MATSim toolkit for doing iterations over initial plans which are being subsequently changed in a "day-to-day" way. We might also call it the "groundhog-day"-replanning, as all agents are doing "one" day all over again, with slightly changed plans for the day.
From the XML-file we load an initial population, represented by several agents who at least hold one plan but might as well have more then one plan. Each agent has one plan that is the “selected plan” and only this plan will be used by the mobsim.
The basic approach is:

1. Read or generate initial plans
2. Iterate
   a. Run mobsim and "observe" the events the mobsim generates.
   b. Feedback the events into a strategic layer
   c. Generate new plans on the basis of the actual plans and the events for a (possibly small) fraction of the population and add those new plans to the populations plans.
   d. Feed the plans into mobsim, where again only the plan marked as selected will be used.
3. End iteration process when a certain condition is met (e.g. number of iterations, near enough to Nash equilibrium)

So what we have to do is set up a strategic layer that is capable to change some or all plans according to the events of the last mobsim-run and some internal rules.

We will build first a manager that handles the distribution of plans to the strategic modules, and then add some of these modules to the manager.
The manager will be created with

```
StrategyModuleManager strategies = new StrategyModuleManager();
strategies.preparePopulation(population);
```

The latter call copes with some initiating stuff for the manager, which we do not care about right now.

Now we can start adding some Modules to the manager. The most basic module one could think about is:

```
strategies.addModule(new KeepLastSelected(), 0.9);
```

This module just does not change any plans or create new ones, instead it just keeps the plan selected, that was already selected.
The second parameter determines the probability the module will be called. What number you choose here is pretty much up to yourself, as the Manager will normalize these values over the sum

of all modules.

So let's add another—more interesting—strategic module:

```
strategies.addModule(new ReRoute(network, events), 0.1);
```

As the name implies, this module chooses new routes for the given plans. In each iteration, 10% of the plans will be randomly selected and handled to this module. Thus, we will re-route 10% of our plans in each iteration and keep 90% of the plans as they are.

If we are finished with adding modules we have to make another call:

```
strategies.connect();
```

which checks all modules for consistency. Why this call is needed and what it is doing will be discussed in more detail in the section about creating your own strategy modules.

To run several iterations, we replace the call to sim.run() with the following loop:

```
for (int ii =0; ii < 99; ii++) {
  events.resetAlgorithms(ii);
  sim.run();
  strategies.runModules(population, ii);
}
```

So we basically just run the simulation over and over again and call the StrategyManager in between to change the population's plans according to the StrategyModules we installed.
The call to `event.resetAlgorithms()` gives every EventHandler the option to prepare for the next run. Although we did not create any EventHandler in our code there is one created implicitly by the ReRoute-module, as the cost function of the Router uses the events to calculate the costs of a route time-dependently.

# Using MATSim with config files

There are two prepared scenarios for using MATSIM with configuration files.
You can use MATSim with an external mobility simulation of your choice, or you can use MATSim with an external re-planning module. Or one could easily use MATSim with the several internal strategy modules.

## How-to write a config file for the internal replanning modules

The easiest way to use MATSim is to use it with the predefined internal replanning modules. The MATSim-Controller class is internally responsible for running several iterations and calling the above mentioned replanning modules. We will use a class called SimpleControler to even easier access the Controler. This SimpleControler class could be given a XML-file – the config-file-- with all the necessary information in it on which plan to use or what replanning modules.

We will see now how such a XML-file has to be constructed.
In the config file (let's say "C:\test\config.xml") you need to specify the following items:
In the module "network" you have to specify an input path to the network-file
E.g.

```
<module name="network">
  <param name="inputNetworkFile" value="C:/test/net.xml" />
```

```
</module>
```

In the module "plans" you have to specify an input path to the (initial) plans-file

```
<module name="plans">
  <param name="inputPlansFile" value="C:/test/plans.xml" />
</module>
```

The last module is for the controlers settings and looks like this:

```
<module name="controler">
  <param name="outputDirectory" value="external/iterations" />
  <param name="firstIteration" value="0" />
  <param name="lastIteration" value="2" />
</module>
```

Here we basically say how many iteration we want to run and where to write the output.
With this simple config file we could start a run with two iterations by this command line:

```
java -cp MATSim_test.jar org.MATSim.examples.with_config.SimpleControler \
          C:\test\config.xml
```

## how to run iterations with built-in modules for plans modification (replanning modules)

We have to add an additional module to our configuration file, which defines the replanning modules which should be used by the iteration-process.
The configuration for the strategy module might look like this:

```
<module name="strategy">

  <!-- ===== Module for selecting the best scored plan ===== -->
  <param name="ModuleProbability_1" value="0.1" />
  <param name="Module_1" value="org.MATSim.demandmodeling.plans.strategies.BestScore"
          />

  <!-- ===== Module for re-routing the plan ===== -->
  <param name="ModuleProbability_2" value="0.1" />
  <param name="Module_2" value="org.MATSim.demandmodeling.plans.strategies.ReRoute"
          />
  <param name="ModuleDisableAfterIteration_2" value="100" />

  <!-- ===== Dummy Module for not touching the plan at all ===== -->
  <param name="ModuleProbability_3" value="0.8" />
  <param name="Module_3" value="KeepLastSelected" />
</module>
```

For each module we want to use we have to give a ModulePropability_X parameter. These parameters will all be normalized with this sum of them all. So here we choose the best scored plan on 10% of the population with Module_1, let 10% of the population choose new routes to their activities with Module_2 and leave 80% of the population unchanged with their last selected plan with Module_3. Further you see that you can disable Modules after a certain number of iterations.

After we added section for plans file, network file and controler settings as above we could start this example with the following command:

```
java -cp MATSim_test.jar org.MATSim.examples.with_config.SimpleControler \
          C:\test\config.xml
```

## How-to do iterations with an external mobsim

Using an external mobsim together with the MATSim-Controler is easy, as long as the external mobsim follows the following conventions:
The external mobsim is called with three parameters from the MATSim Controler:

```
./external_mobsim config-file plans-file events-file
```

where:

config-file is the full path to the config file used to start the mobsim
plans-file is the full path to the XML-Plans file
events-file is the full path to where the application should write the events-file to

The mobsim does not need to be able to access the config file, as the two important entries from the config file (where to get the plans from and where to put the events to) are already given by parameter 2 and 3, but we highly recommend to parse the config file instead and to put any needed information for your mobsim into an additional module-section, so you have all information for one simulation iteration in one distinctive file.

You have to give the path to a temporary directory, where the (input-)plansfile will be written to and the path to your mobsim application in the module "simulation". The filenames are fixed as "external_mobsimplans.in.xml" and "external_mobsimevents.out.xml"
E.g.

```
<param name="ExternalMobSimTmpFileRootDir" value="C:/temp/external" />
<param name="ExternalMobSimExePath" value="C:/test/mobsim.exe" />
```

Your mobsim will be called now like this:

```
C:/test/mobsim.exe C:\test\config.xml C:/temp/external/external_mobsimplans.in.xml
          C:/temp/external/external_mobsimevents.out.xml
```

You can start this example with the following command:

```
java -cp MATSim_test.jar org.MATSim.examples.with_config.ExternalMobsimOnce
          C:\test\config.xml
```

If you want to run iterations with your mobsim you will need to use another java class and you will have to specify some more variables:

Start the iterating version with the following command:

```
java -cp MATSim_test.jar org.MATSim.examples.with_config.ExternalMobsim
          C:\test\config.xml
```

For a specification of the XML events that need to be written refer to "http://matsim.org/wiki" under "XML-Events"

## running an external module for plans modification (planomat.exe)

*This example has been written with the external module Planomat in mind an right now is basically only useful for integrating the planomat.exe into MATSim iterations. Will will expand this to more general approach soon!*

This case is a little bit more complicated than just running with the internal modules. The module for replanning now looks like this:

```
<module name="strategy">
  <param name="ExternalExeConfigTemplate"
         value="external/config_template_planomat.xml" />
  <param name="ExternalExeTmpFileRootDir" value="external/" />

  <!-- re-route is not used as a strategy in this setup, but needed for rerouting the
         planomat files!! -->
  <param name="ModuleProbability_1" value="0.0" />
  <param name="Module_1" value="org.MATSim.demandmodeling.plans.strategies.ReRoute"
         />

  <!-- planomat is used for all persons -->
  <param name="ModuleProbability_2" value="1.0" />
  <param name="Module_2"
         value="org.MATSim.demandmodeling.plans.strategies.PlanomatExe" />
  <param name="ModuleExePath_2" value="C:/test/planomat/planomat.exe" />

  <!-- ===== Dummy Module for not touching the plan at all ===== -->
  <param name="ModuleProbability_3" value="0.0" />
  <param name="Module_3" value="KeepLastSelected" />
</module>
```

In the example given, three different strategy modules are used.
`Module_1` is not assigned any probability, so it is never actually used on it's own. But it is used to fulfill a constraint from the planomat module, which needs re-routing of the changed plans. As planomat rearranges the departure times of our plans it is necessary to re-route our plans according to the changed departure times.

`Module_2` is the actual external module. Here we use planomat. We can specify the location of the external module with `ModuleExePath_2`. Further we need to specify an usable config file for the external exe, that holds all modules important to the external replanning module. In case of planomat that should be the following modules:

```
<module name="root_dir">
<module name="scenario">
<module name="planomat">
<module name="ga">
<module name="utility">
<module name="XMLPlansIdentifiers">
```

In the scenario-module there will be two parameters automatically set to the right values:

```
<param name="inputPlansFilename" value="planomat_plans.in.xml"/>
<param name="workingPlansFilename" value="planomat_plans.out.xml"/>
```

These values point to the plans written by the Controler and to a place where the updated plans should be written by planomat.

`Module_3` is also not used. If it would be used, it would leave the plans unchanged and doing nothing. So if you just want to send some part of the plans to planomat you have to change the module-probability-values.
Appending the following lines in our module "strategy" would lead to only 20% of the plans being re-planned by planomat:

```
<param name="ModuleProbability_2" value="0.2" />
```

```
<param name="ModuleProbability_3" value="0.8" />
```

The external strategy module—in this case planomat—is given only one parameter, which is the place where this config-file could be found.

In the end, the call to the strategy-module would look like this (the config file is this time a modified copy of the original config file, with updated parameters, it will be therefore found in the controllers "tmp" path!)

```
C:/test/planomat/planomat.exe C:/temp/external/ctrl/tmp/planomat_config.xml
```

Again the last module is for the controlers settings and looks like this:

```
<module name="controler">
      <param name="outputDirectory" value="external/iterations" />
      <param name="firstIteration" value="0" />
      <param name="lastIteration" value="2" />
</module>
```

Calling the iterations with the external replanning-module from MATSim could be tested with the follwing call:

```
java -cp MATSim_test.jar org.MATSim.examples.with_config.ExternalReplanning \
          C:\test\config.xml
```