

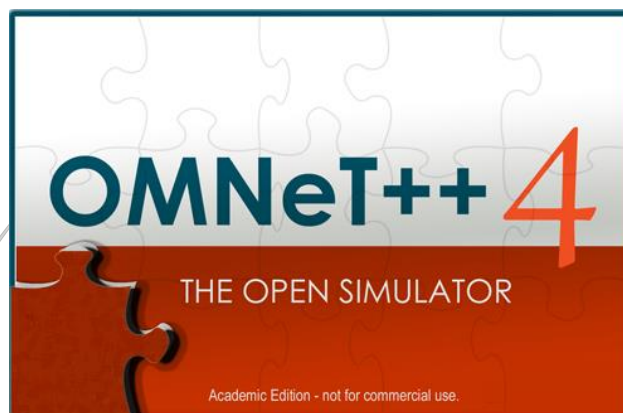


1/1/2015



Traffic Generator Implementation with Omnet++

OMNET++ Workshop



Mohamed Elshaikh

SCHOOL OF COMPUTER AND COMMUNICATION ENGINEERING,
UNIMAP

Objectives

- Omnet++ files Architecture and types
- Protocol files (cc)
- Protocol parameters files (ned)
- Scenario parameters file (ini)

Introduction

The term traffic generator is widely used in the network simulation tools. It refers to a module for generating packet or data to be sent for a destination/destinations. The main aim of this generator is to simulate the real world traffic that could be generated by the network users. A traffic generator module is a statistical model for the usage of the network. It is also a model of the data follow in the network. For example in a network the packets or data that created by the end user of the network is modeled as a traffic generator. Traffic generation could be varying based-on the user, or network's application; for example the traffic generated y an email user is not the same as multimedia user of the same network.

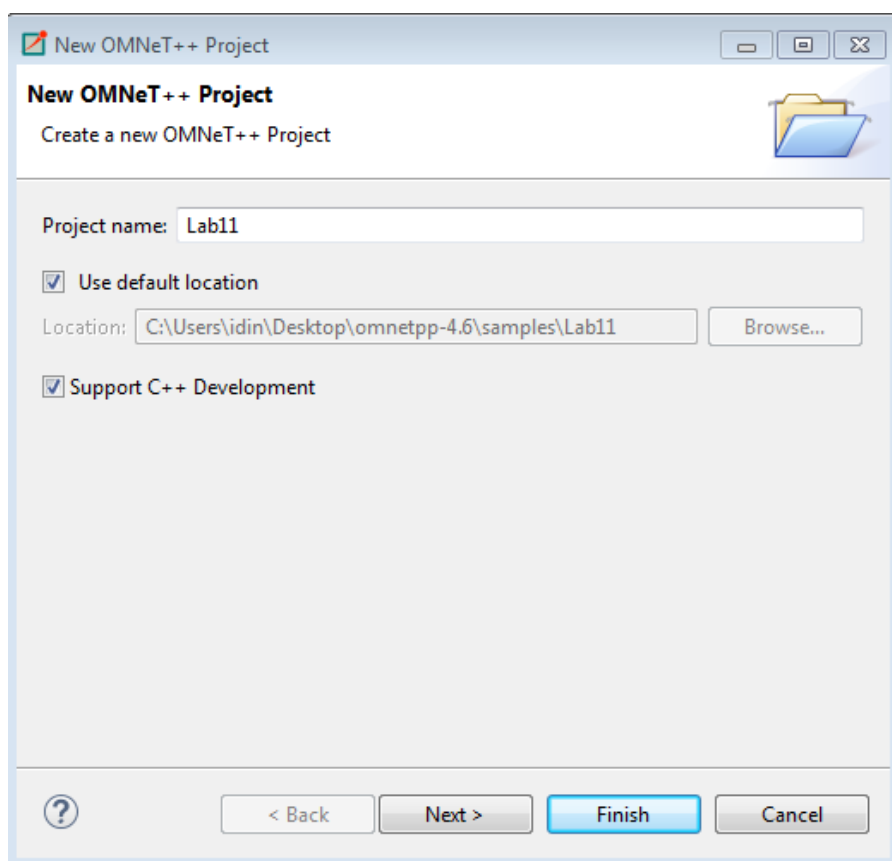
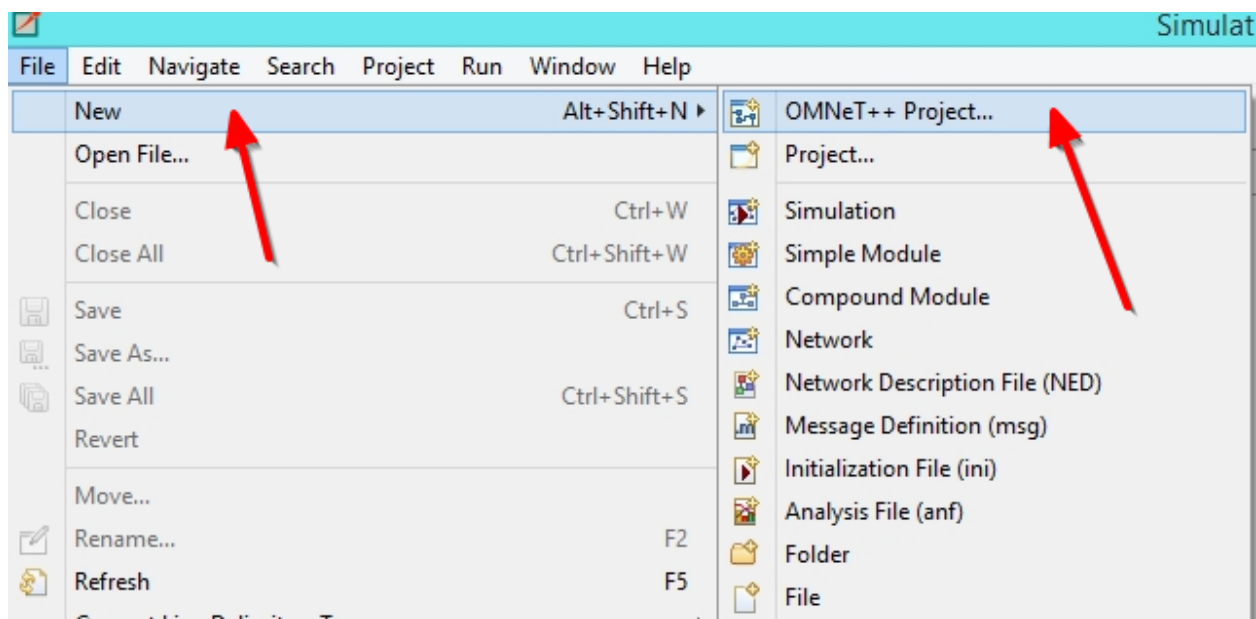
Example 1.

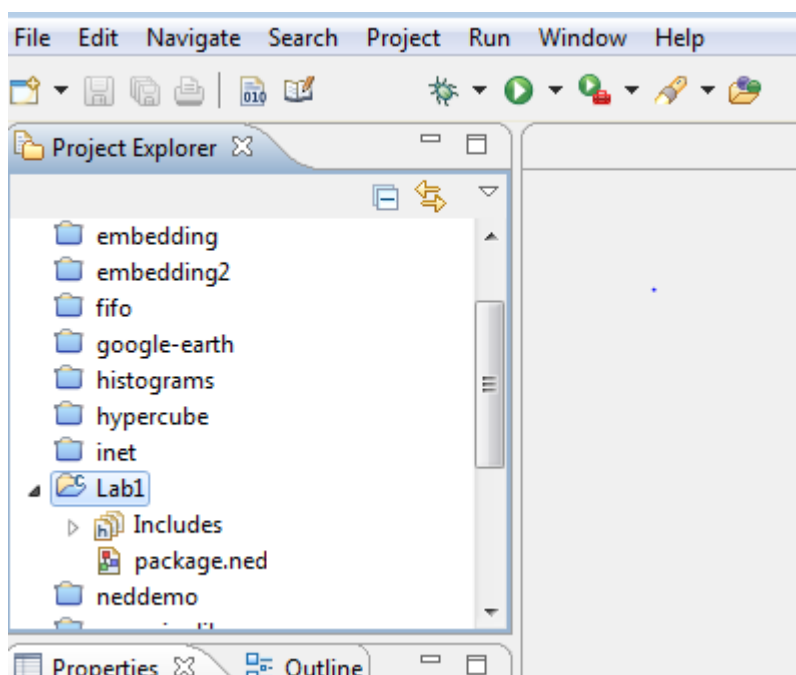
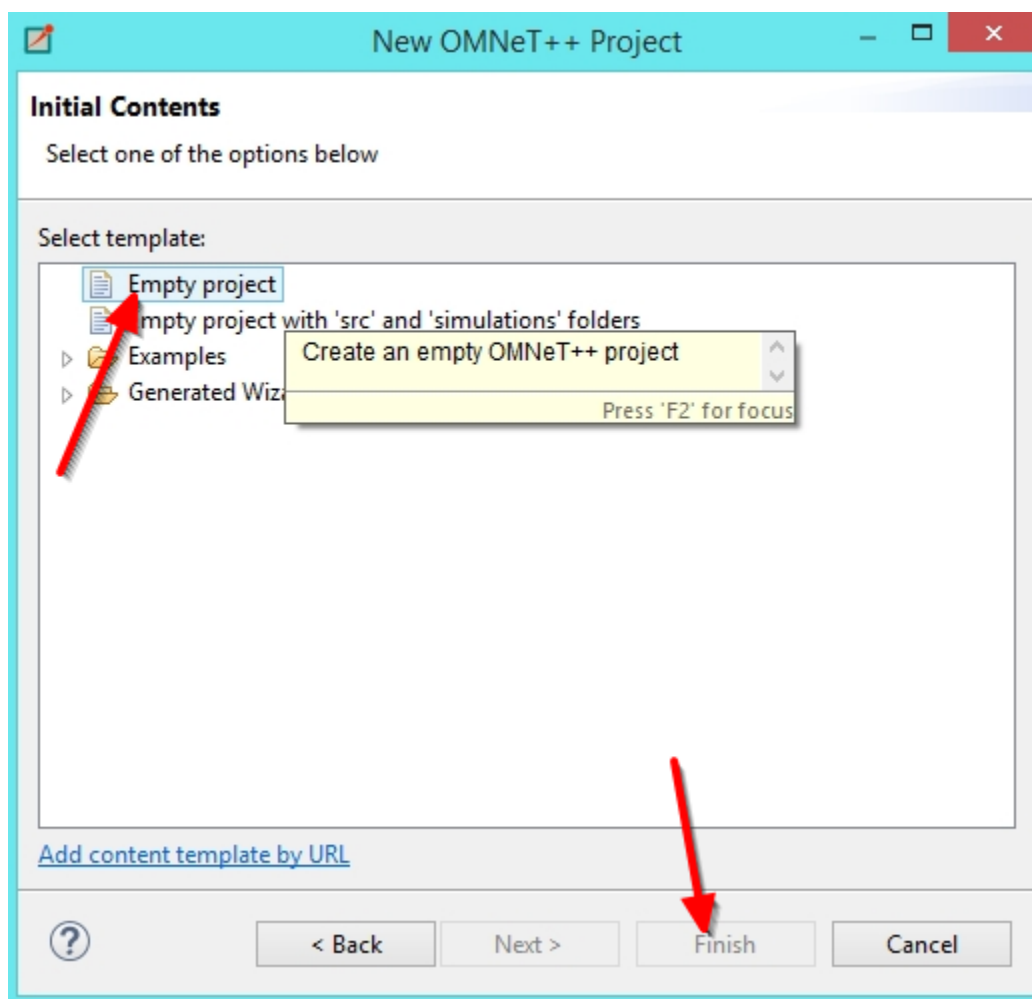
A simple traffic generator. The aim of this example is to build a simple traffic generator. The traffic generator (Gen) is to generate traffic according to two parameters (time interval and packet size). In omnet++ a protocol implementation should consist of at least two file (source file Gen.cc and network description file Gen.ned). the source file is a C++ implementation of the protocol mechanism and the ned file works as a GUI for the source file (used for changing values of some protocol parameters "time interval and packet size"). The sections leads us through the implementation of the example.

Step 1. New project in omnet++

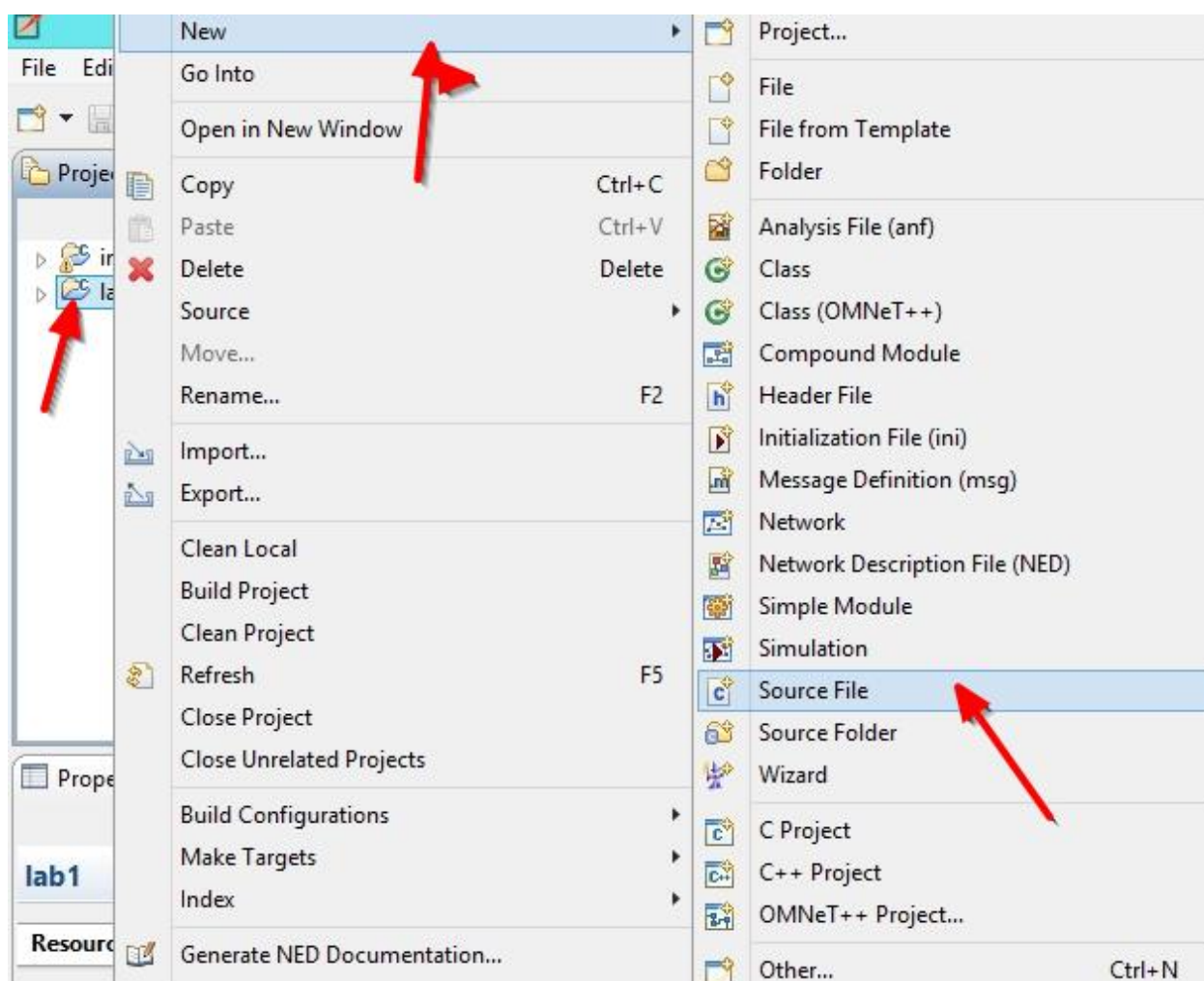
To create a new project in omnet++

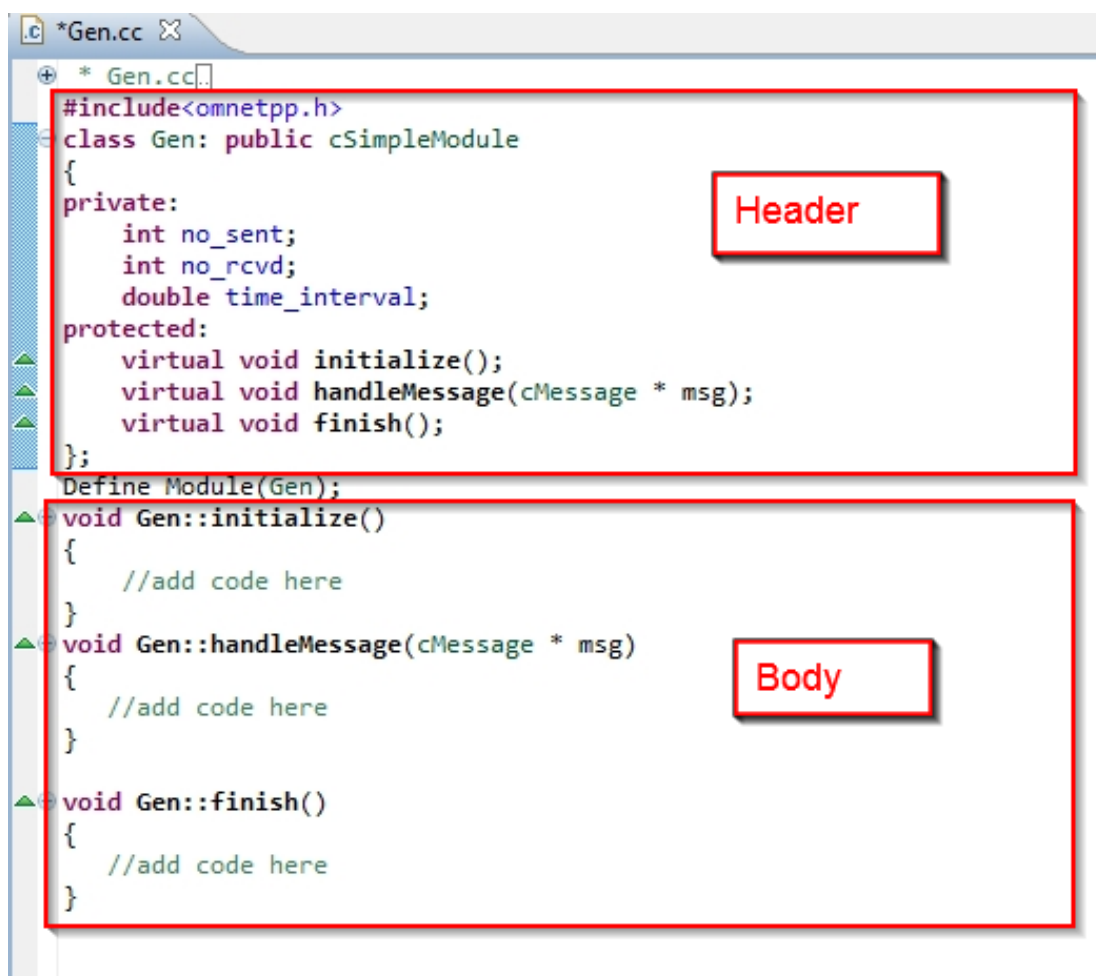
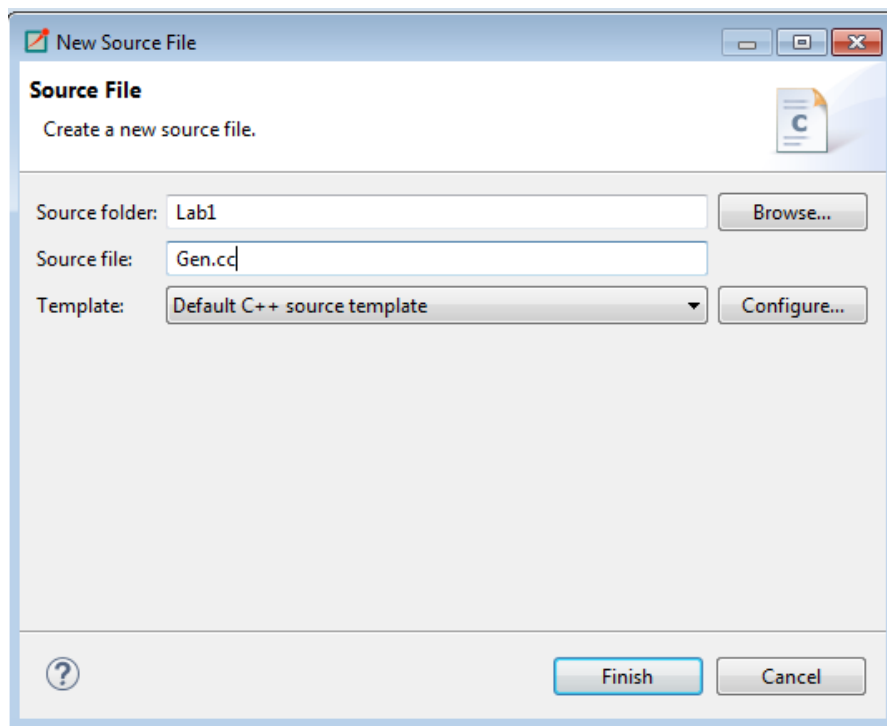
1. Open omnet++ from toolbar menu create New **Omnet++ Project** and name it Lab1, in the next dialog chose **empty project** and click **finish**.





2. From the **folder Lab1** right click → **New** → **Source File** and Name it **Gen.cc**, and write this code in the **Gen Class**





```
#include<omnetpp.h>
class Gen: public cSimpleModule
{
private:
    int no_sent;
    int no_rcvd;
    double time_interval;

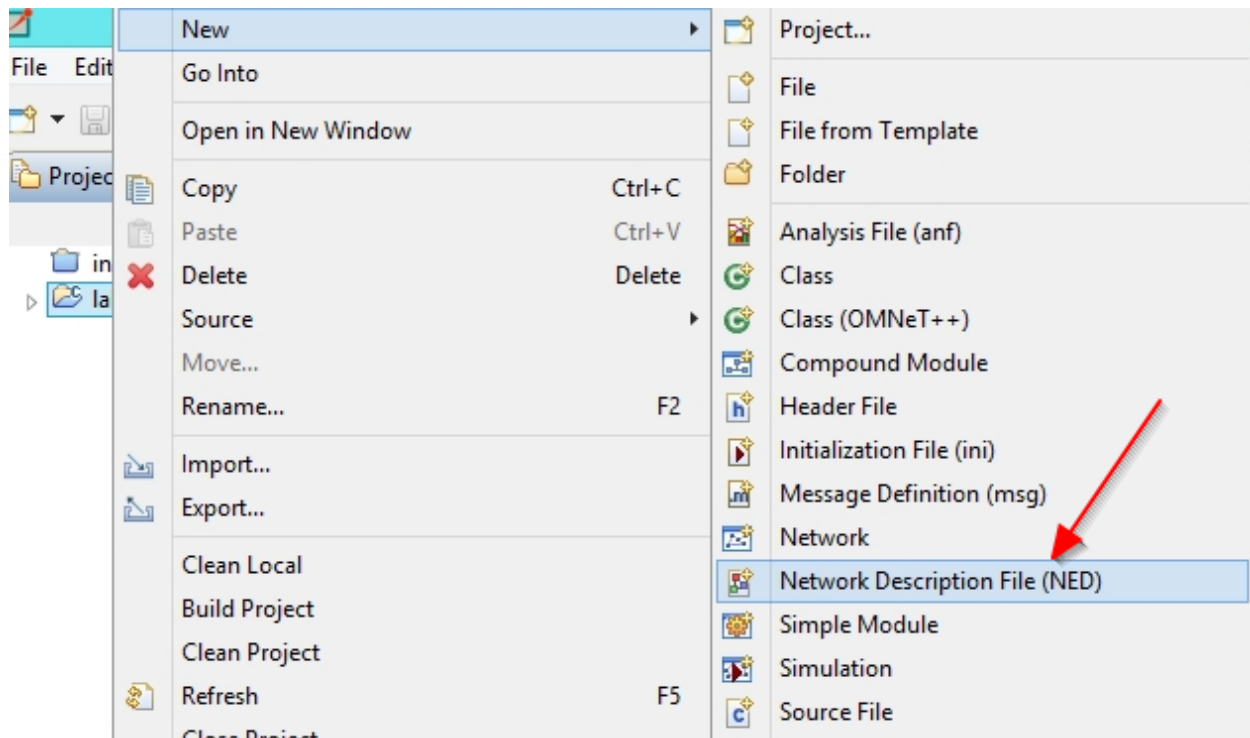
protected:
    virtual void initialize();
    virtual void handleMessage(cMessage * msg);
    virtual void finish();
};
Define_Module(Gen);
void Gen::initialize()
{
    no_sent= 0;
    no_rcvd= 0;
    time_interval= 0.1;
    cMessage *msg= new cMessage();
    scheduleAt(0.01,msg);
}
void Gen::handleMessage(cMessage * msg)
{
    if(msg->isSelfMessage())
    {
        cMessage * out_msg= new cMessage();
        send(out_msg,"out");
        no_sent++;
        scheduleAt(simTime()+time_interval,msg);
    }
    else
    {
        no_rcvd++;
        delete(msg);
    }
}

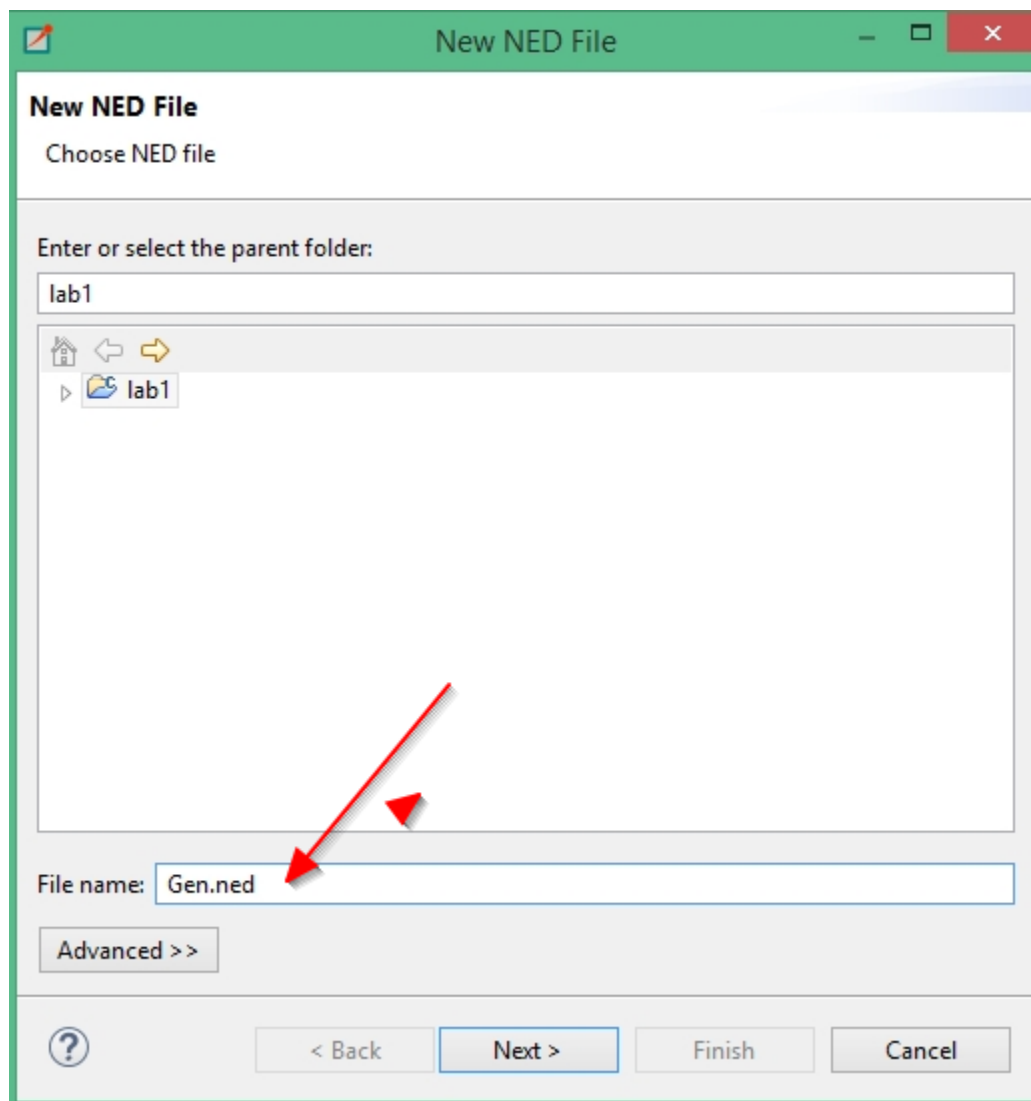
void Gen::finish()
{
    recordScalar("NUmberof received messages",no_rcvd);
    recordScalar("Number of sent messages",no_sent);
}
```

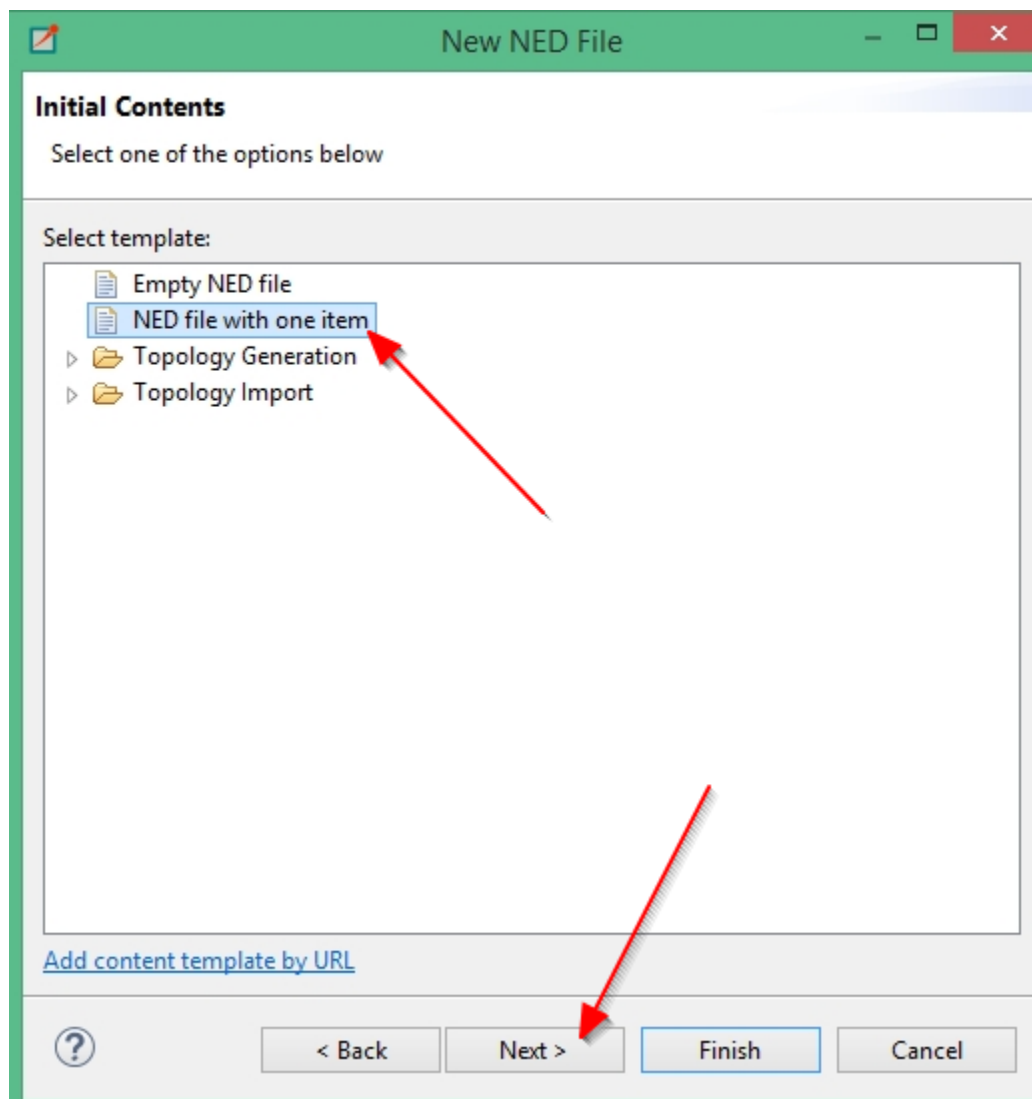
After finish use **ctr+B** to build the class (make sure no errors (**red only**))

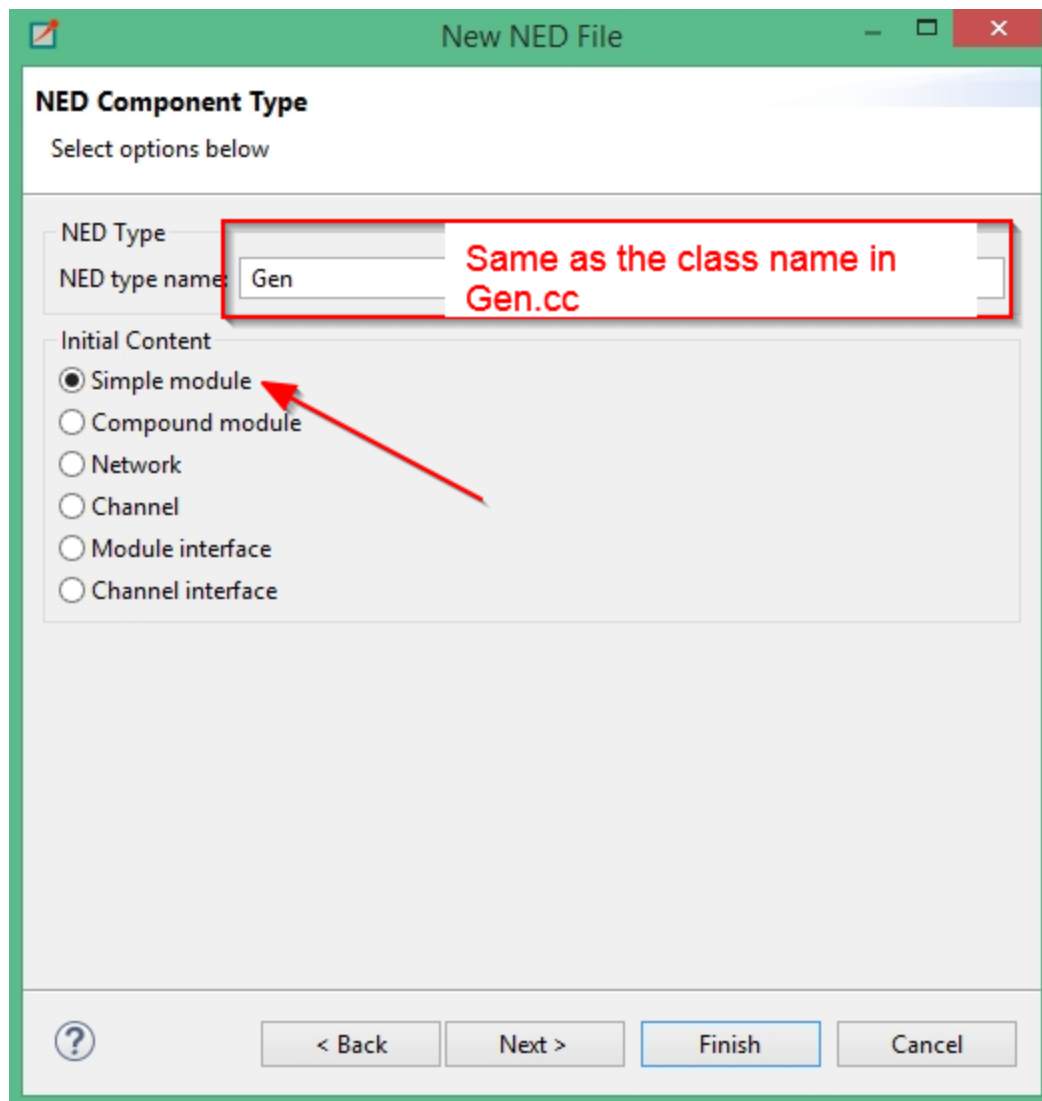
B. Protocol Parameters file (.ned)

1. Right click on the project folder and new->network Description file(NED). And name it Gen.ned. in the next dialog choose ned with one item and next. In the third dialog tick on the simple module choice and finish (**make sure the name is same as the class name in the Gen.cc file**).

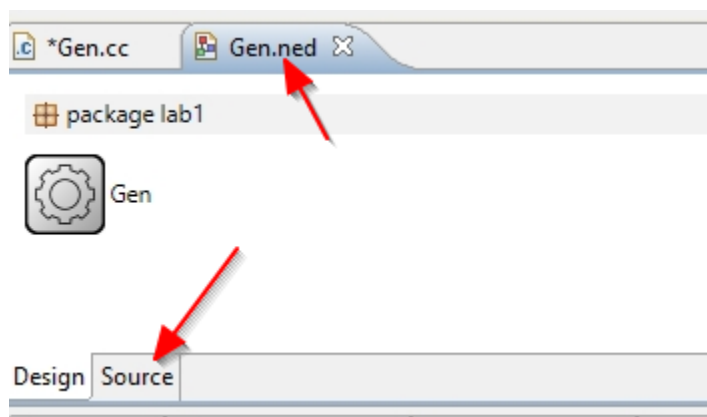


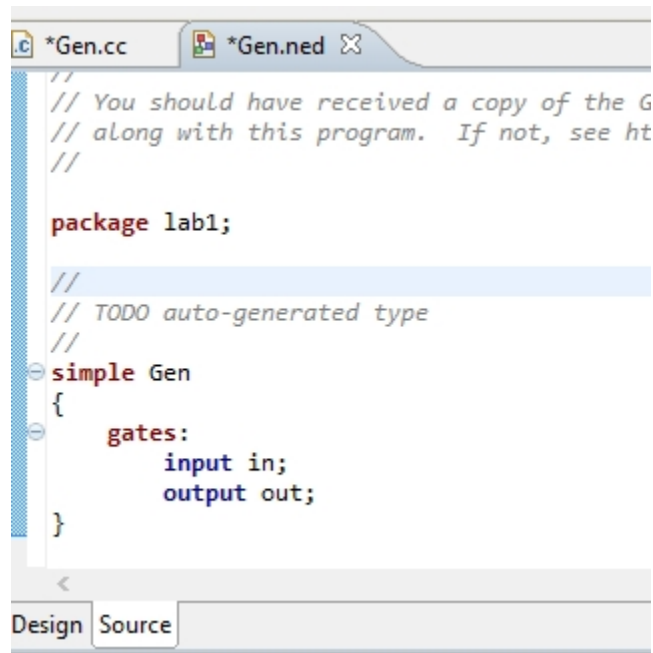






Switch to Source tab and add the following code:

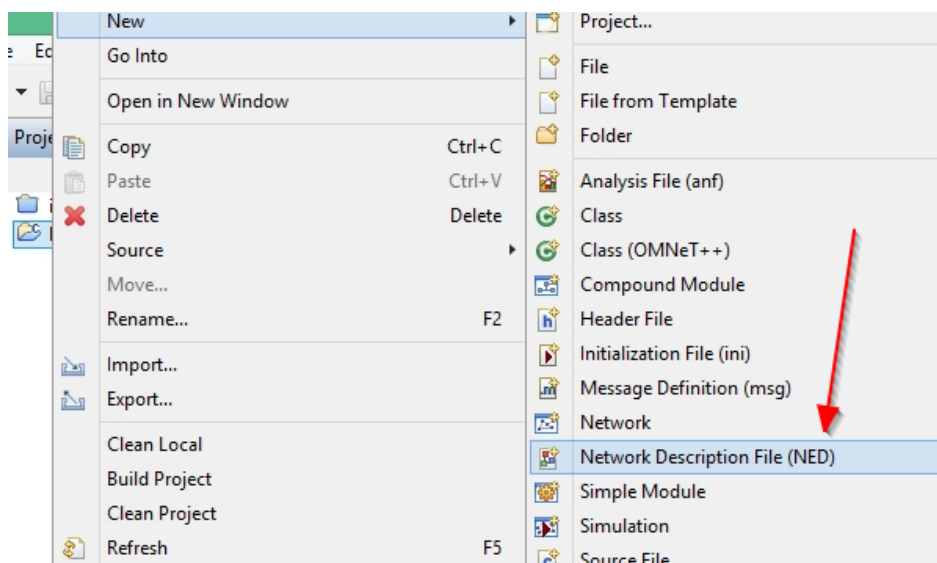




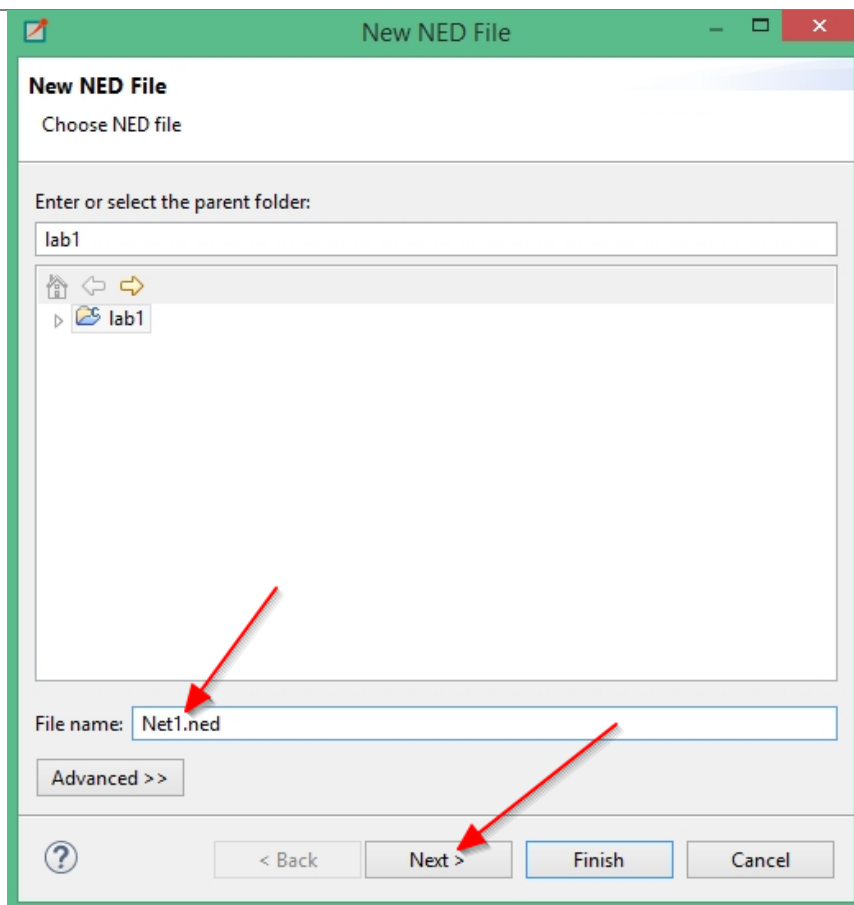
Now we have our generator is ready to use. The next step we build a network with our generator and start run it.

2. To create a network it's Same like create **Gen.ned**, but in the third dialog we should tick on **network** Not simple module and name it **Net1.ned**

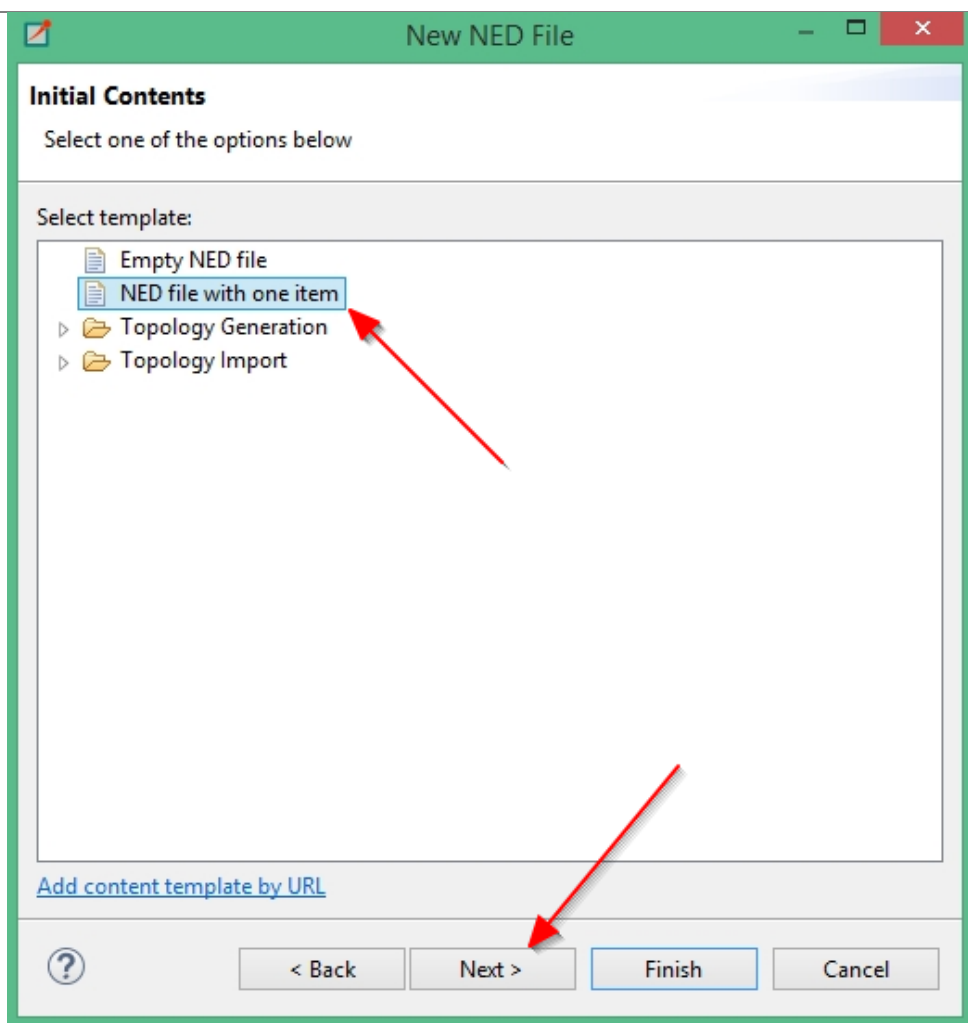
1



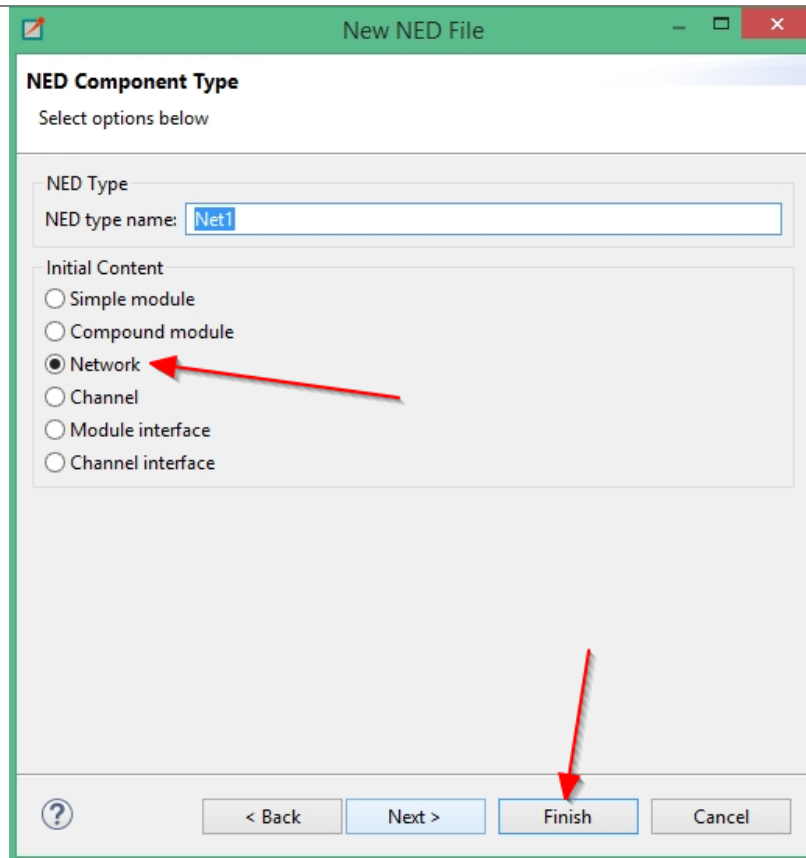
2



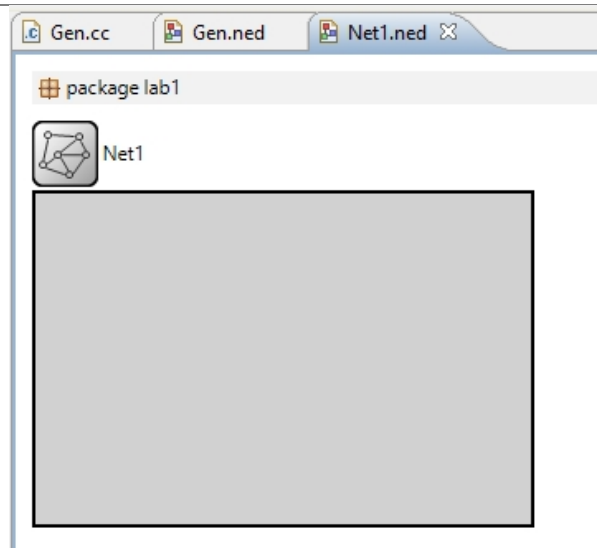
3



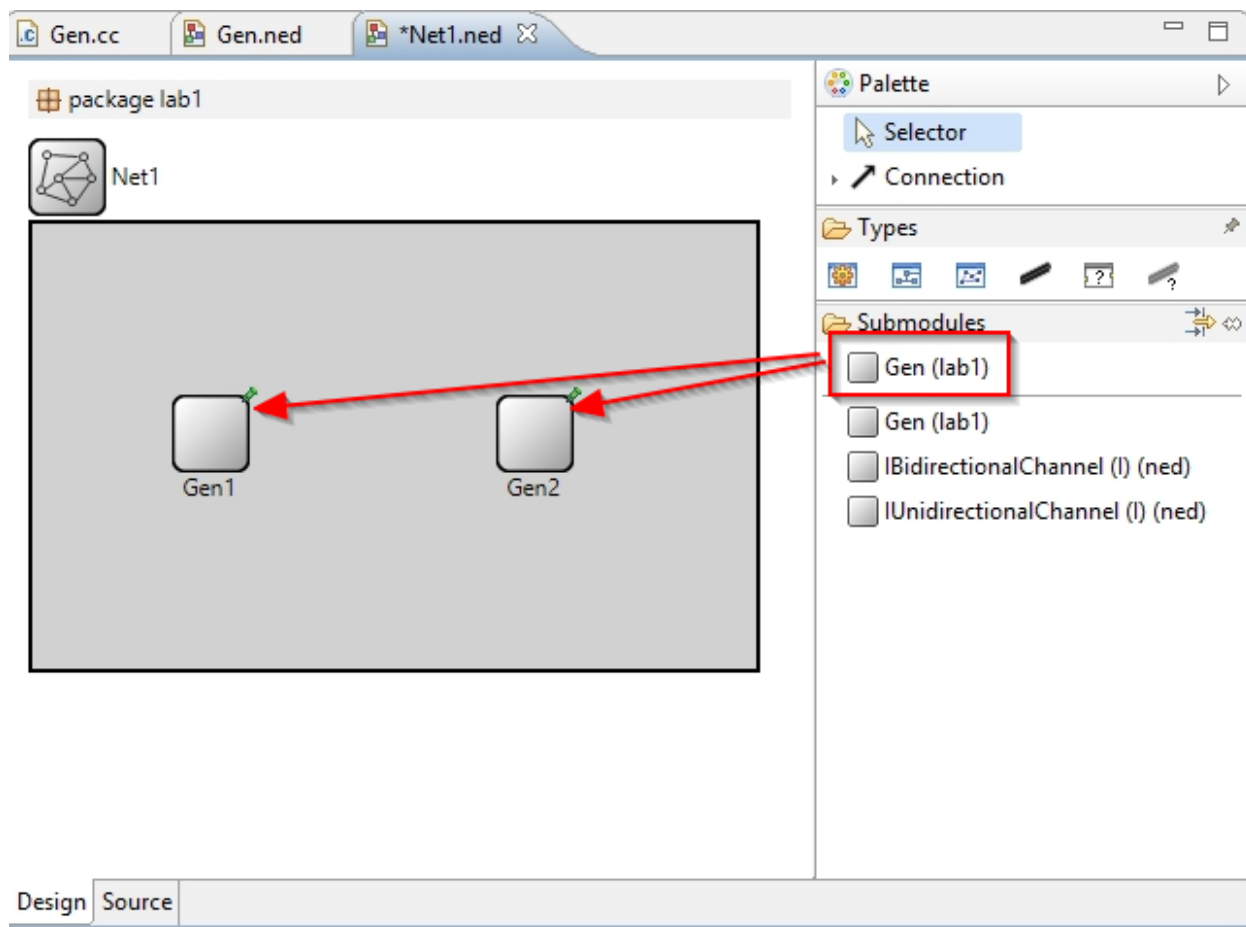
4



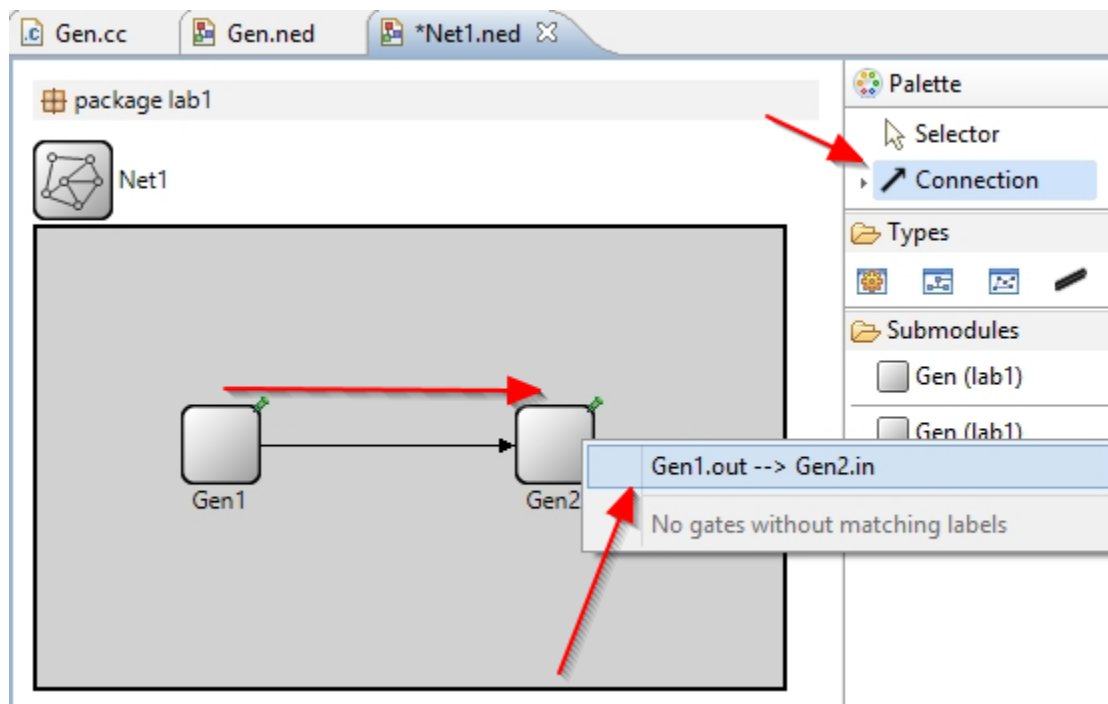
5



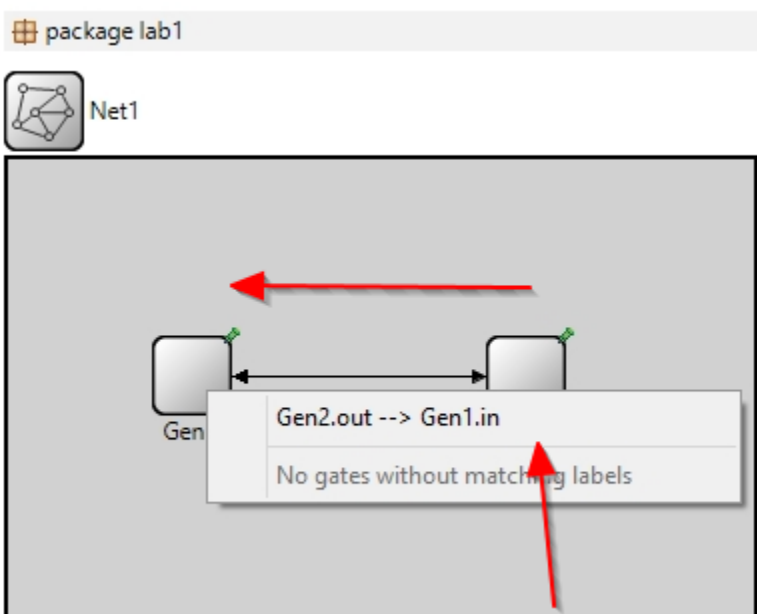
On the network add two Generators from the Platte on the right side

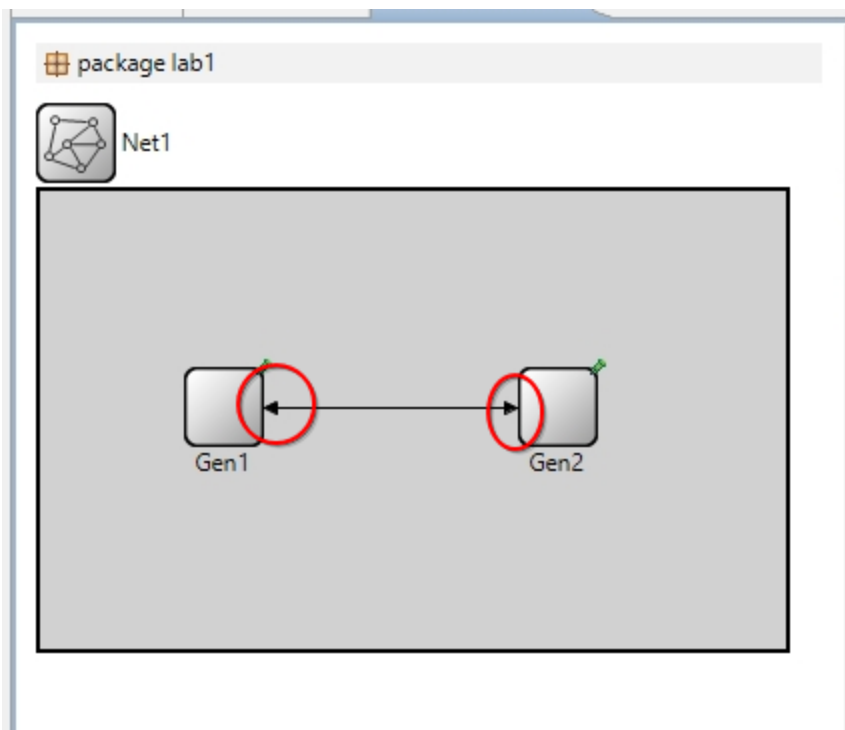


Connect the nodes with the connection (two way connection) tool from the palette (in to out and out to in)



And



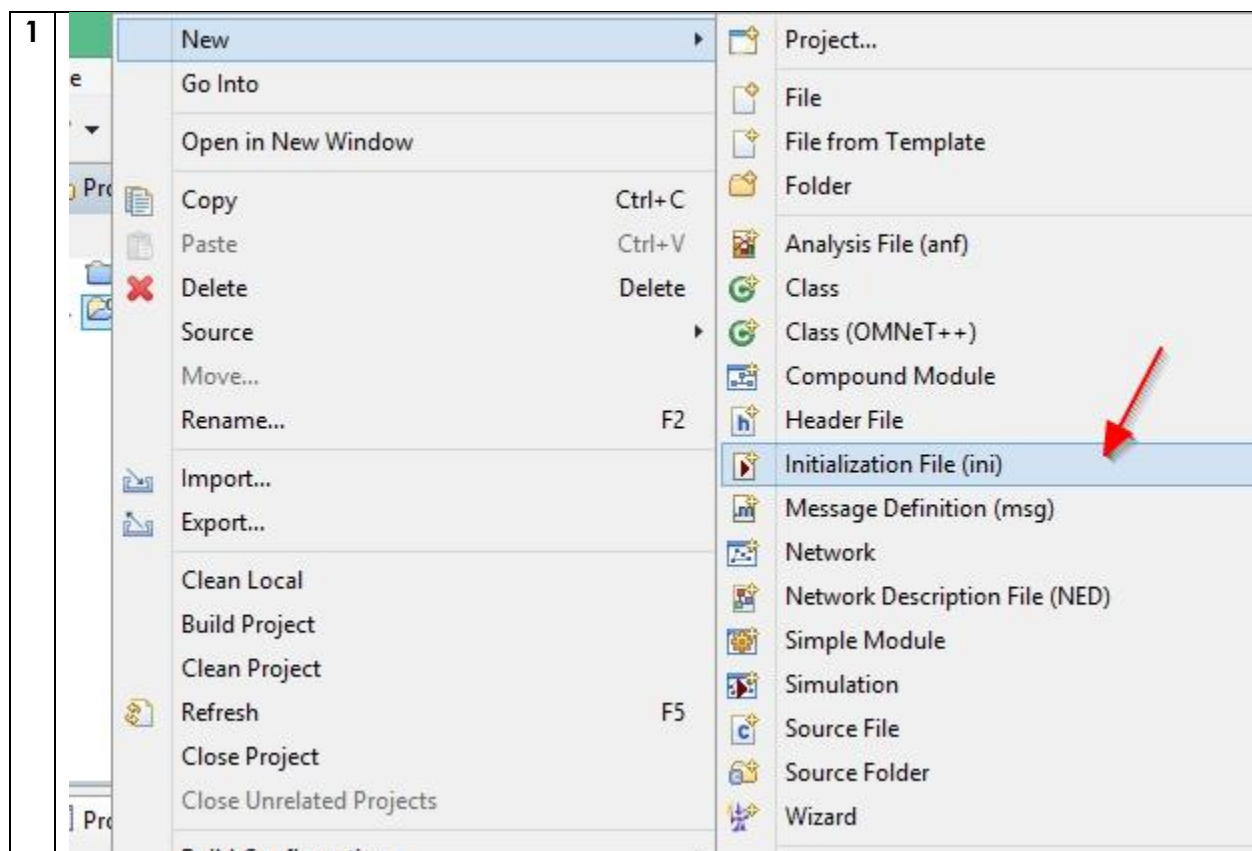


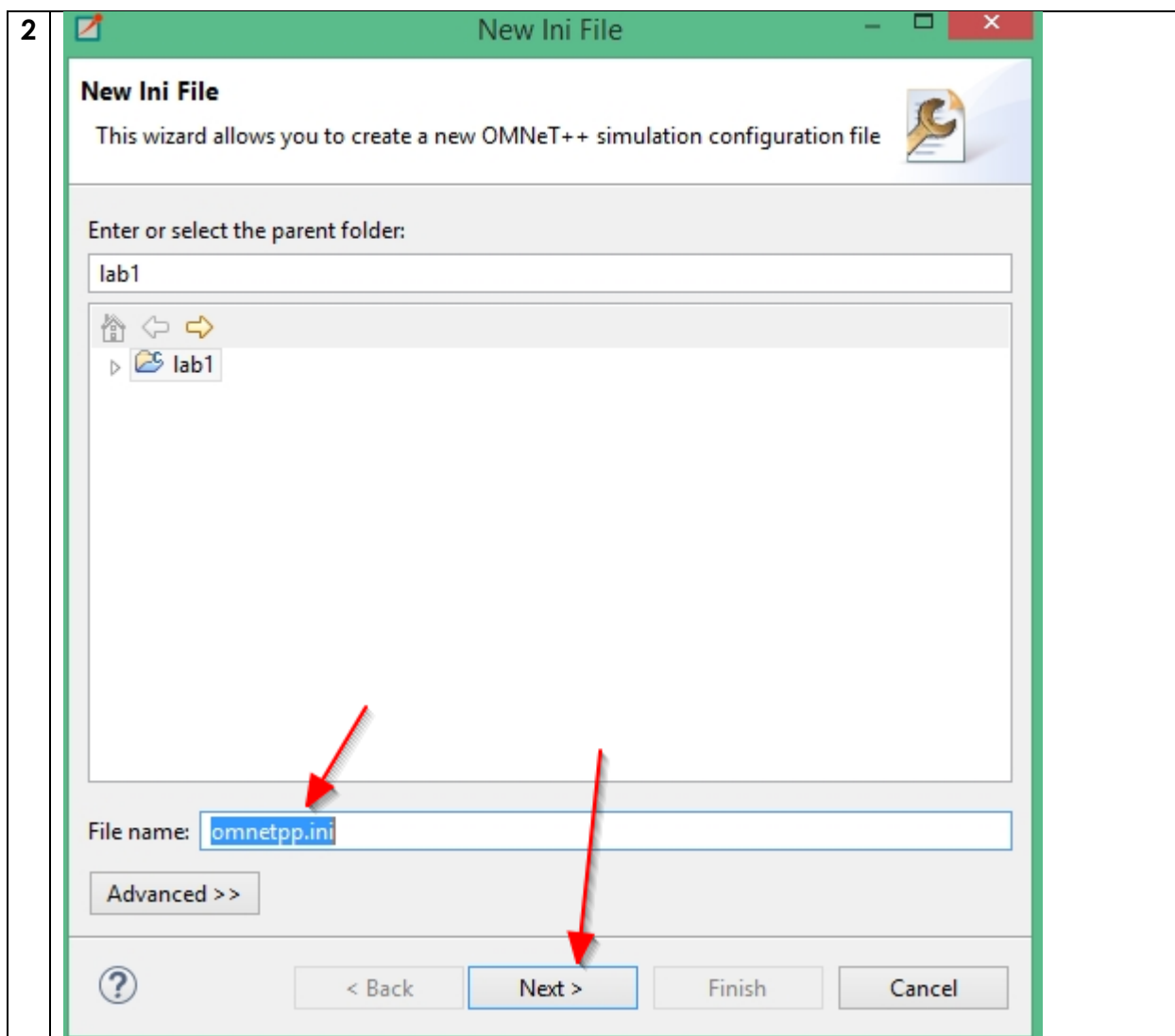
Now the network is ready to run

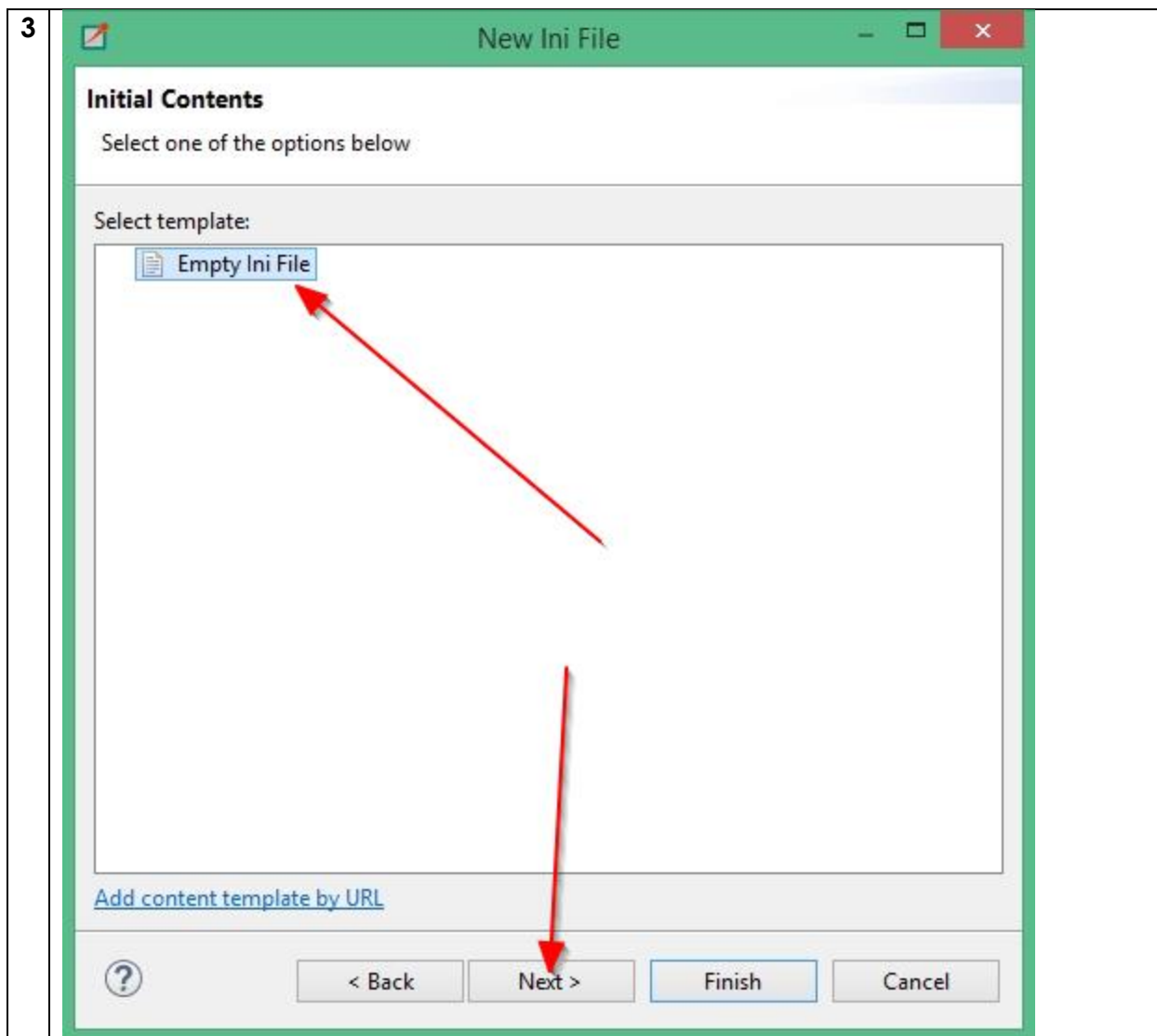
C. Scenario Parameters File (.ini)

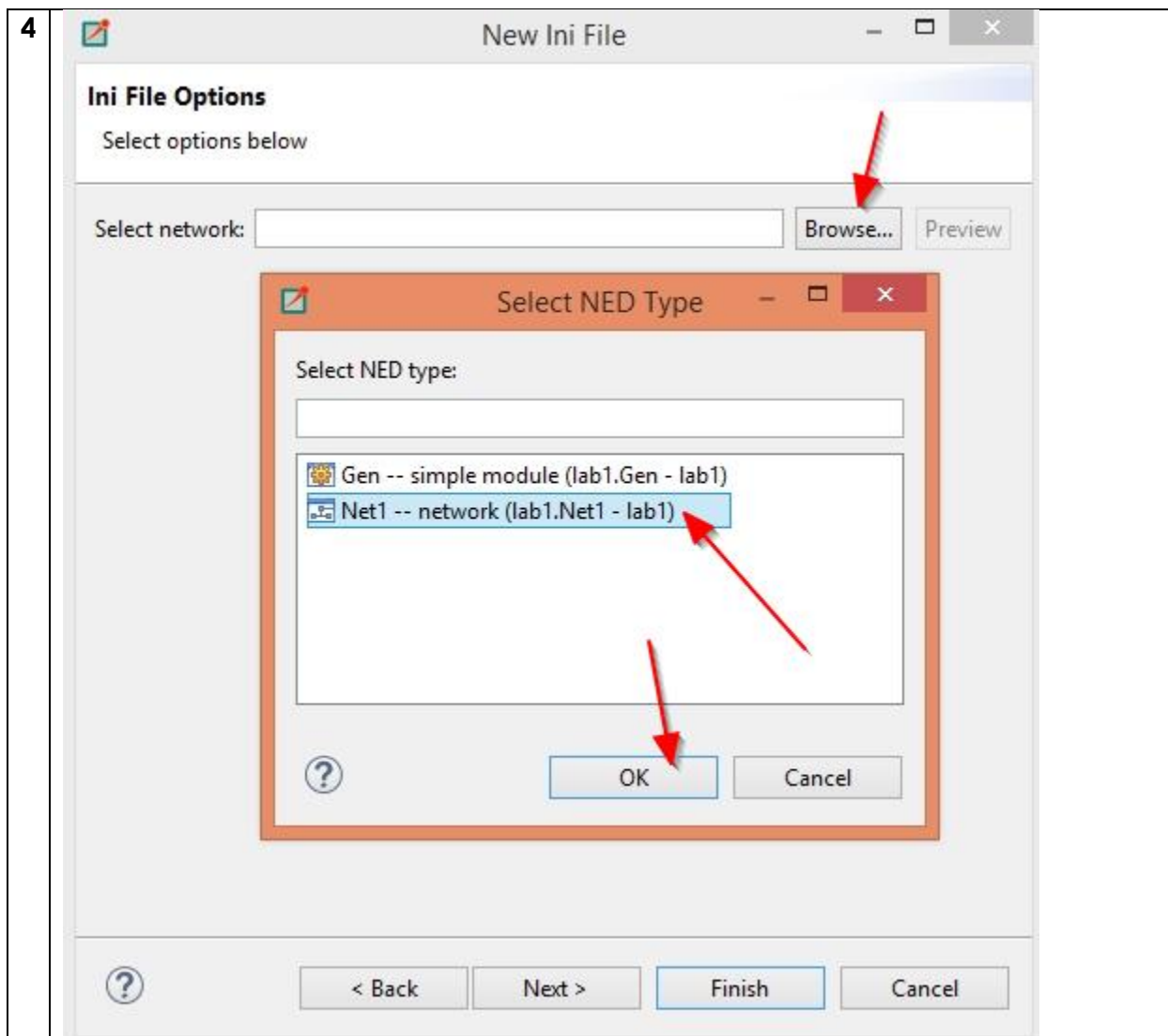
Before we could run this scenario we should create the ini (initialization file),

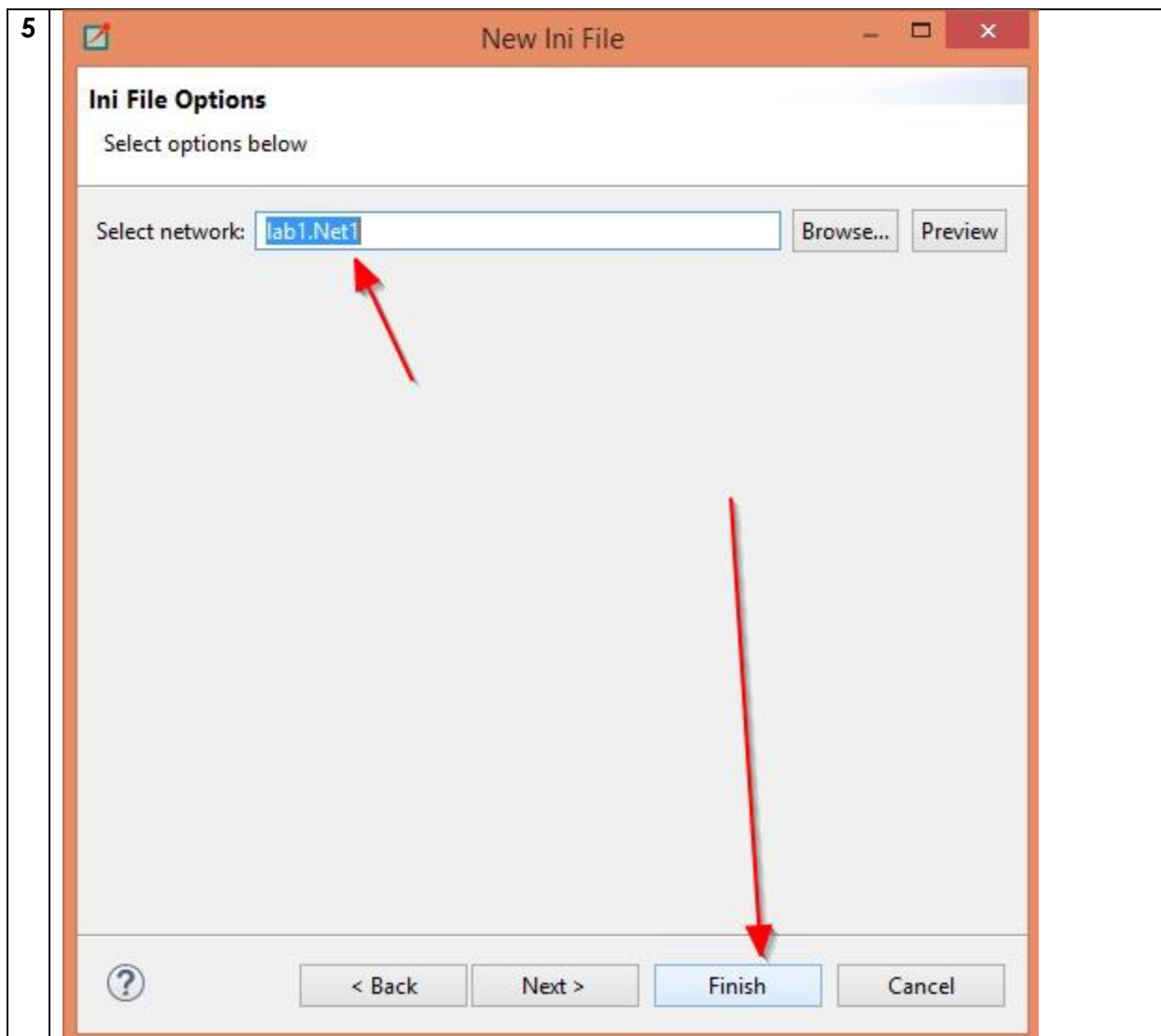
Create **Omnetpp.ini**

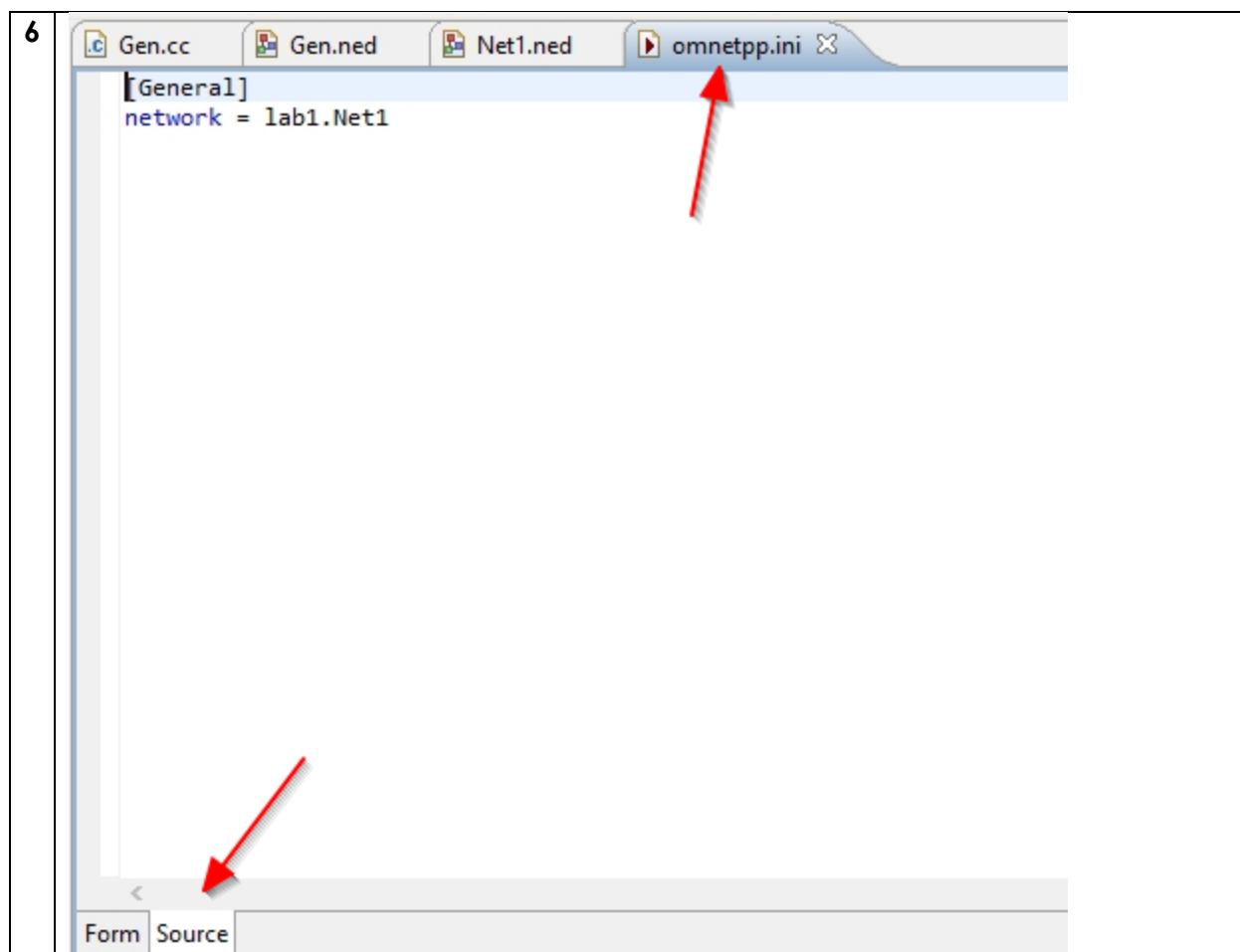




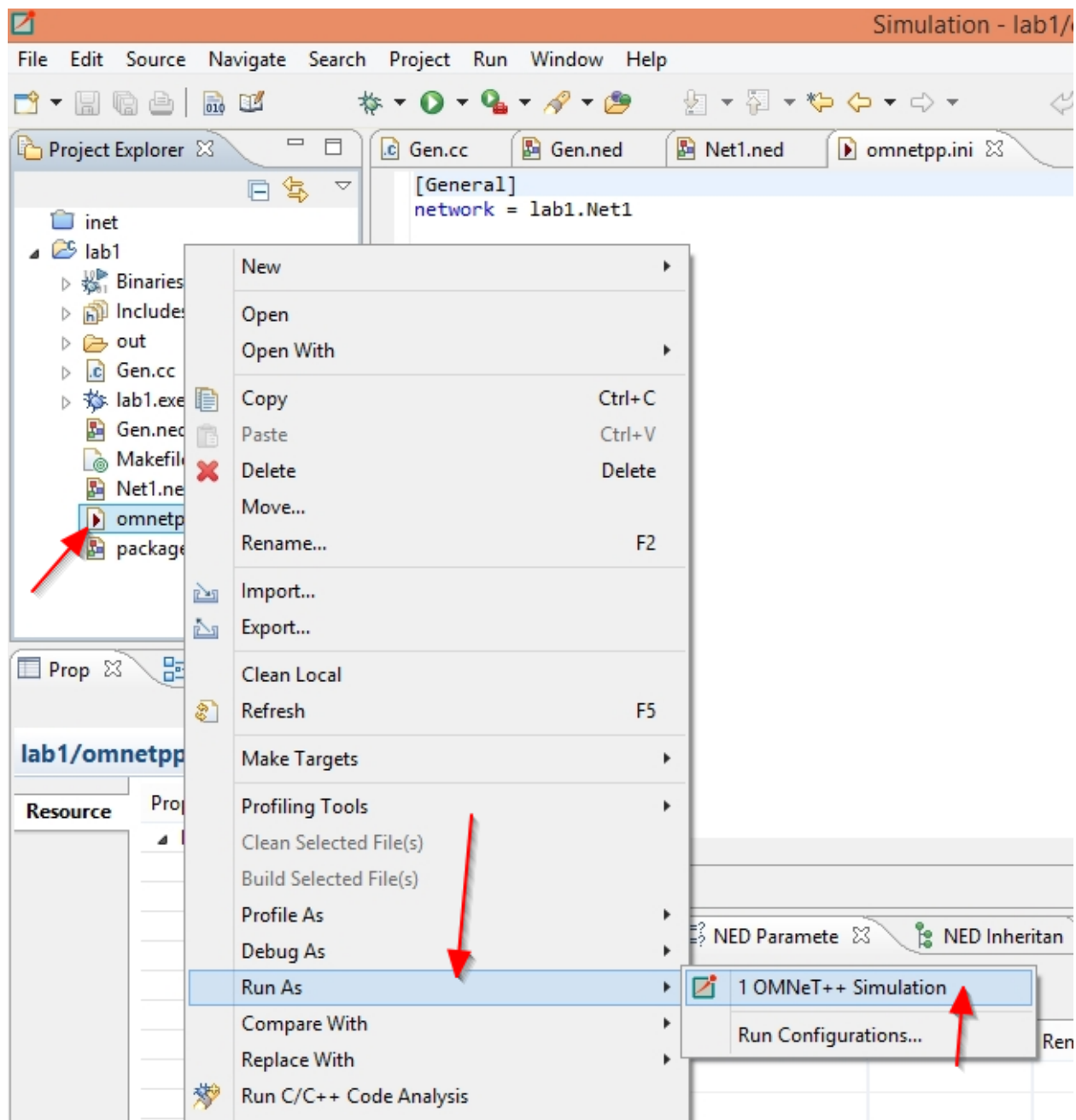


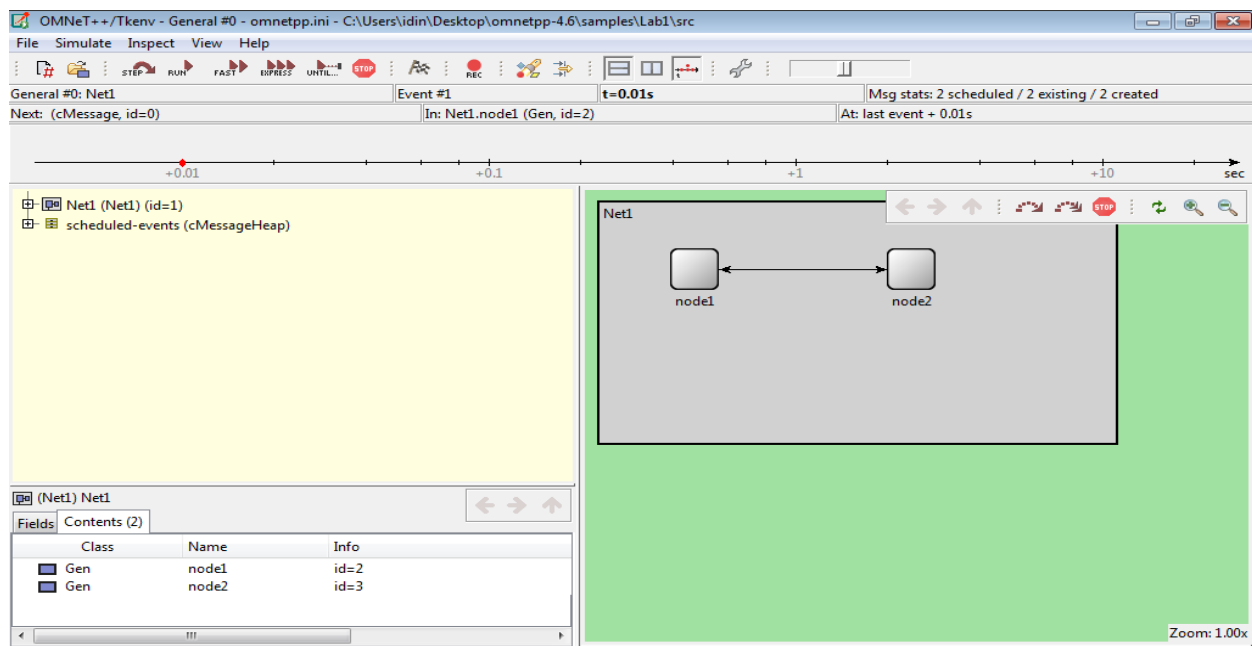




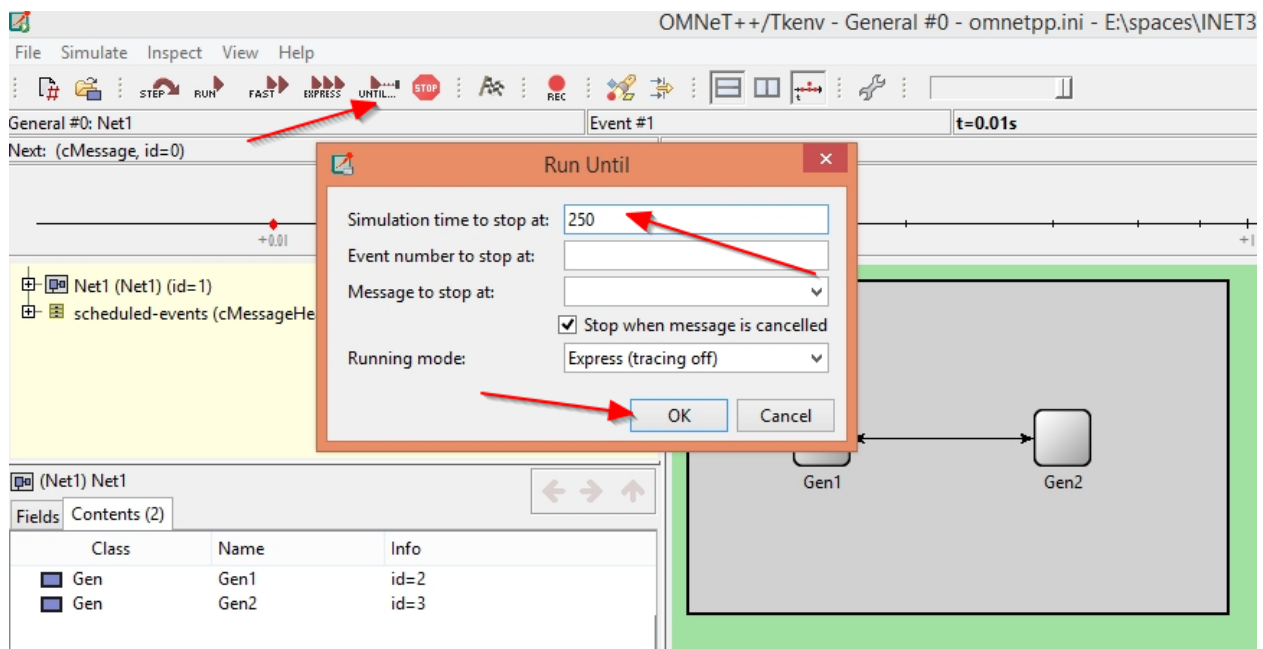


Now we can run our scenario as :

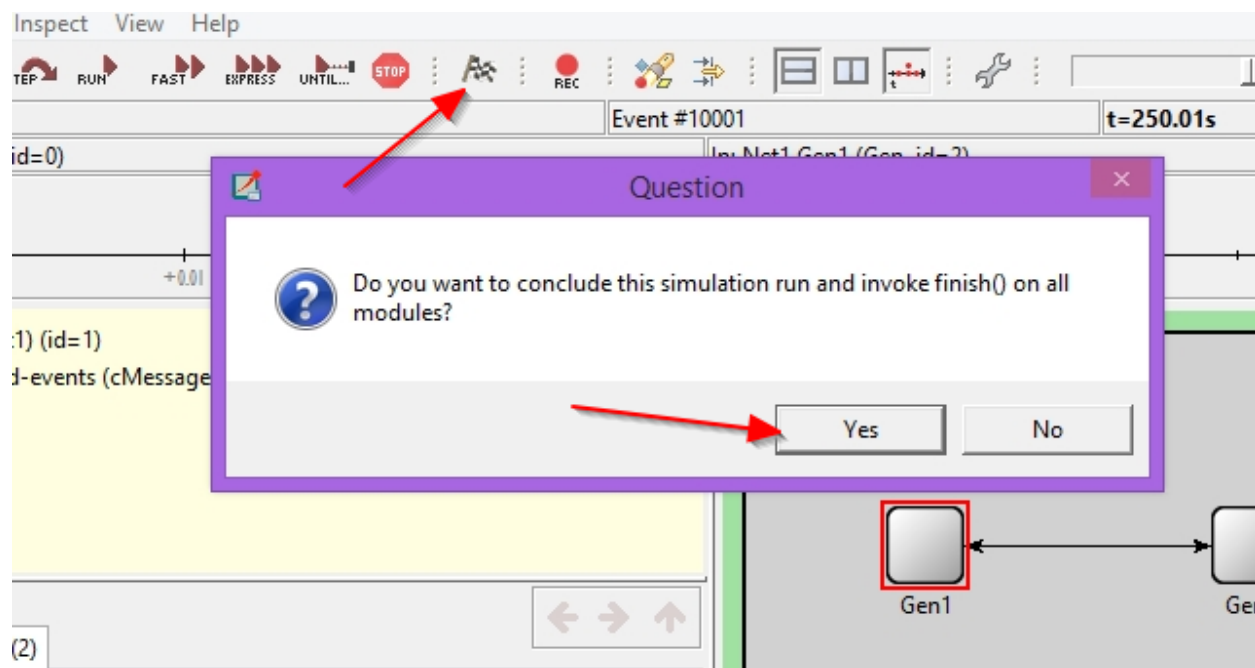




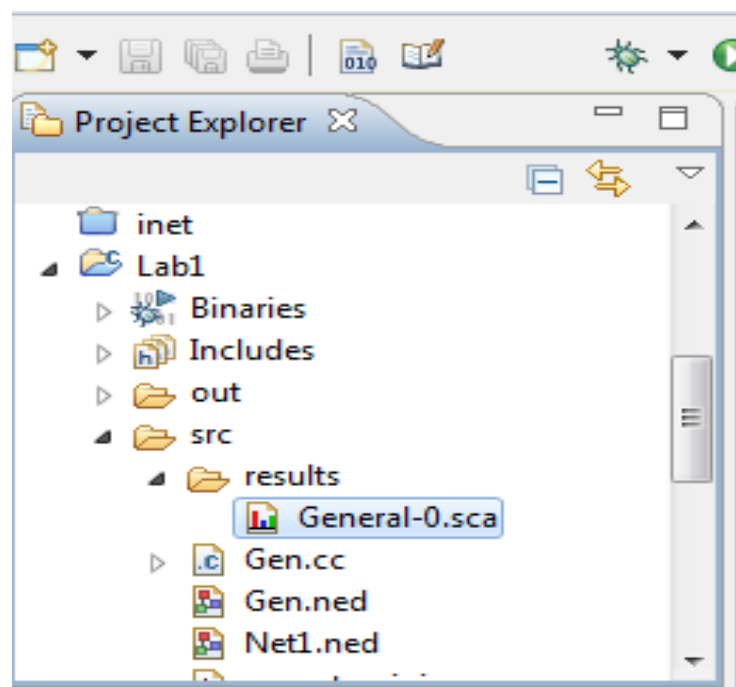
Press **run until tool**, and run the simulation for 250 seconds.



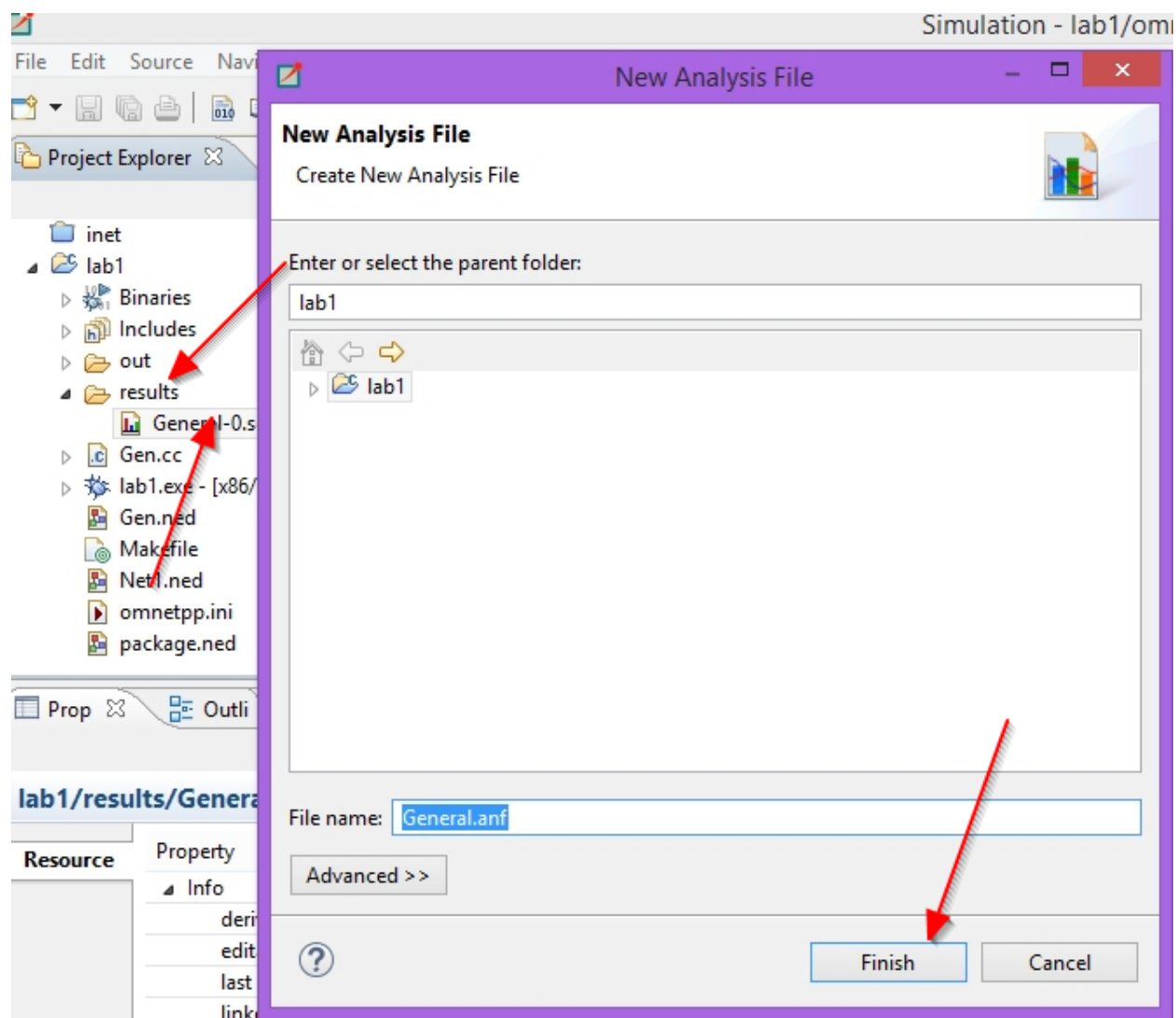
Call finish and close



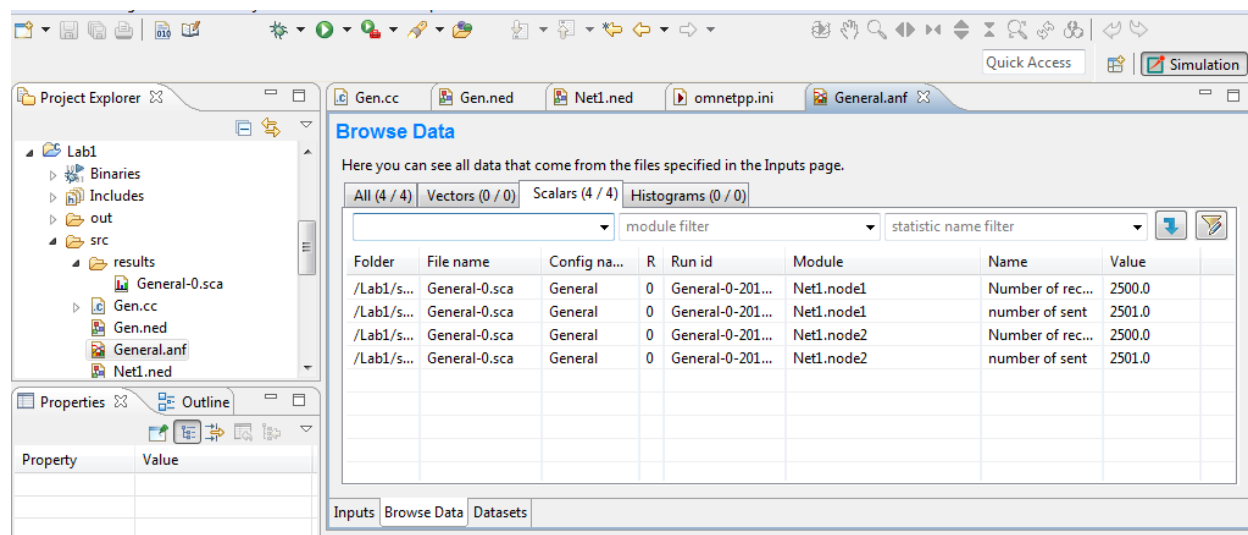
A new folder with the name result will be created under the project folder. Under the result folder open the file (General-0.sca)



Open the general-0.sca to create (auto generated) general.anf file



Shift to browse sheet and observe our results on the scalar tab.



Example 2.

Changing parameters and values from the ini file.

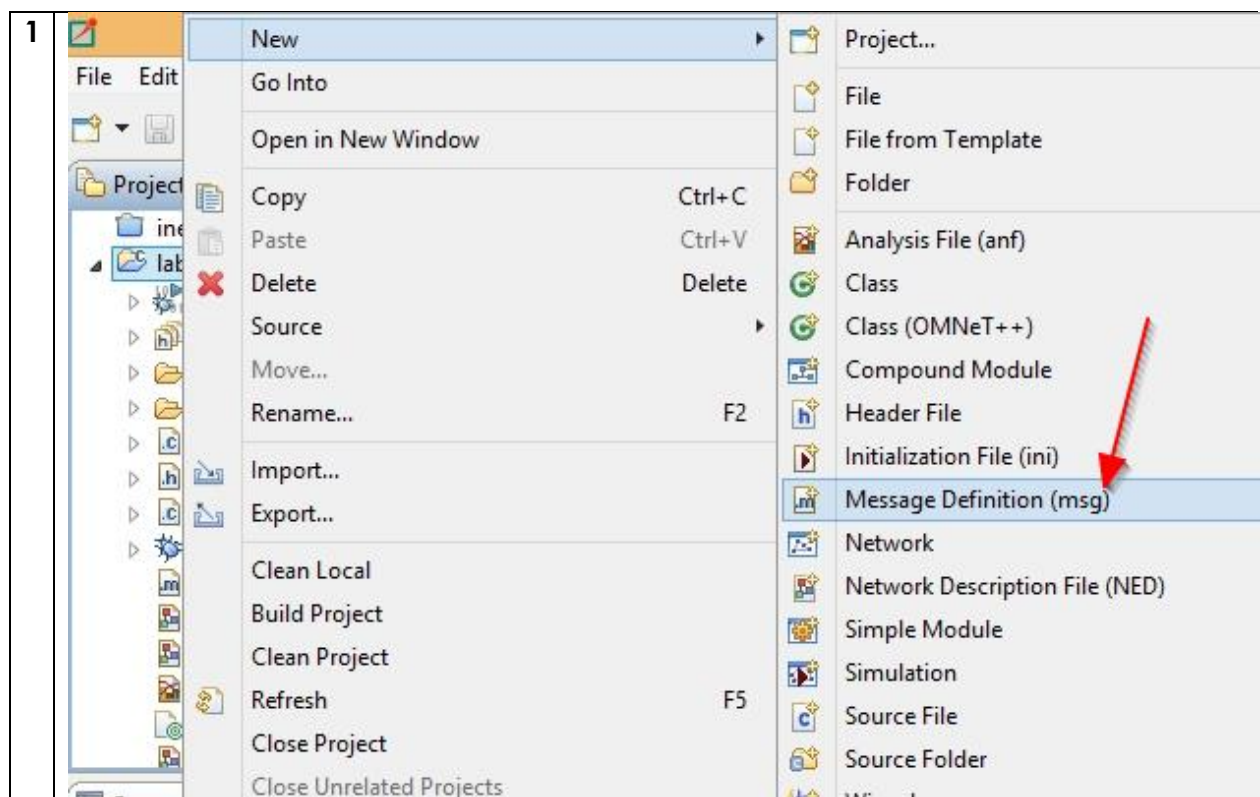
In this example we will:

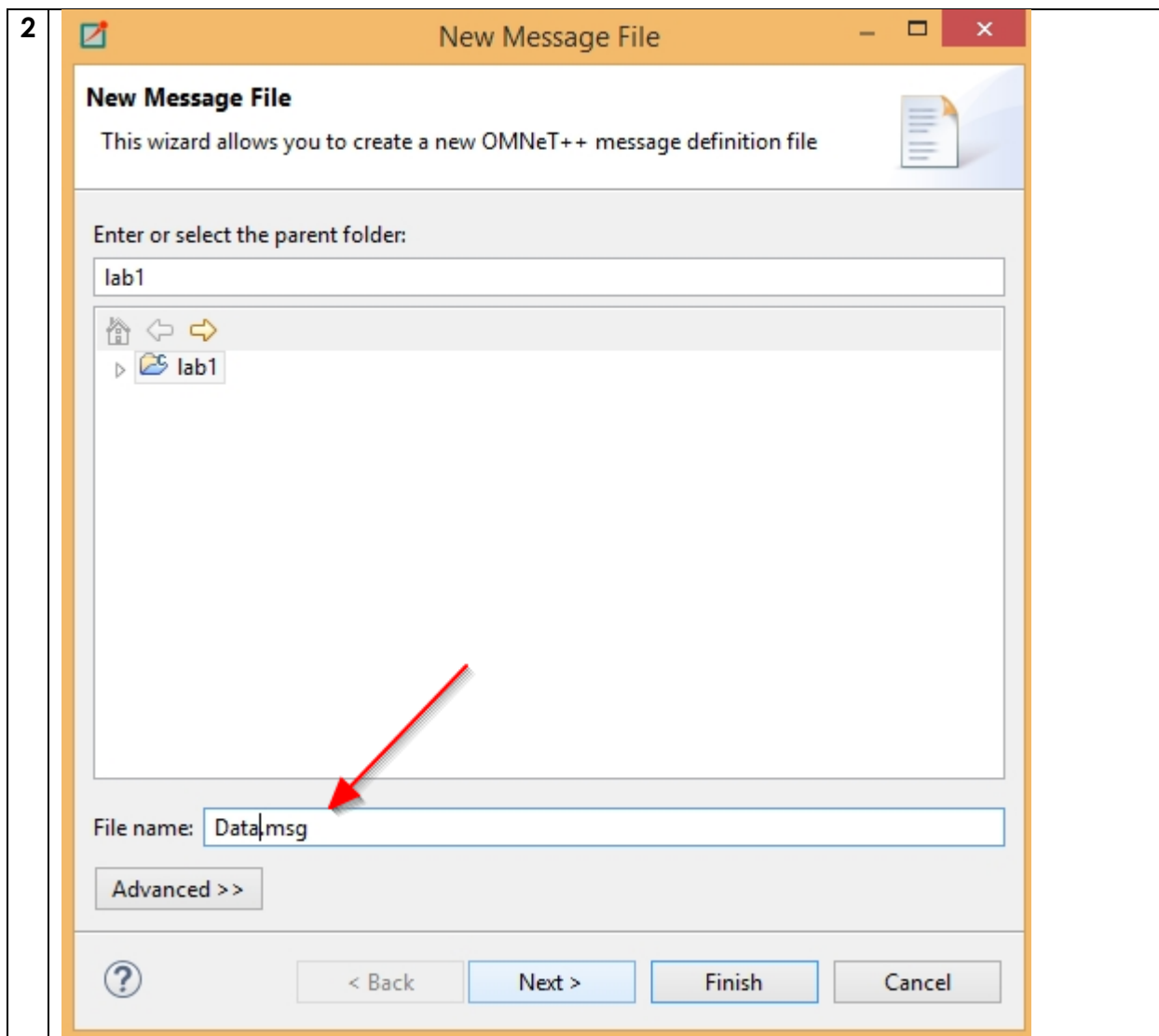
- 1- Link the time interval value in the last example to be changed every time we run the simulation.
- 2- Add some delay to link between nodes
- 3- Packet definition
- 4- Add statistic results (delay vector)

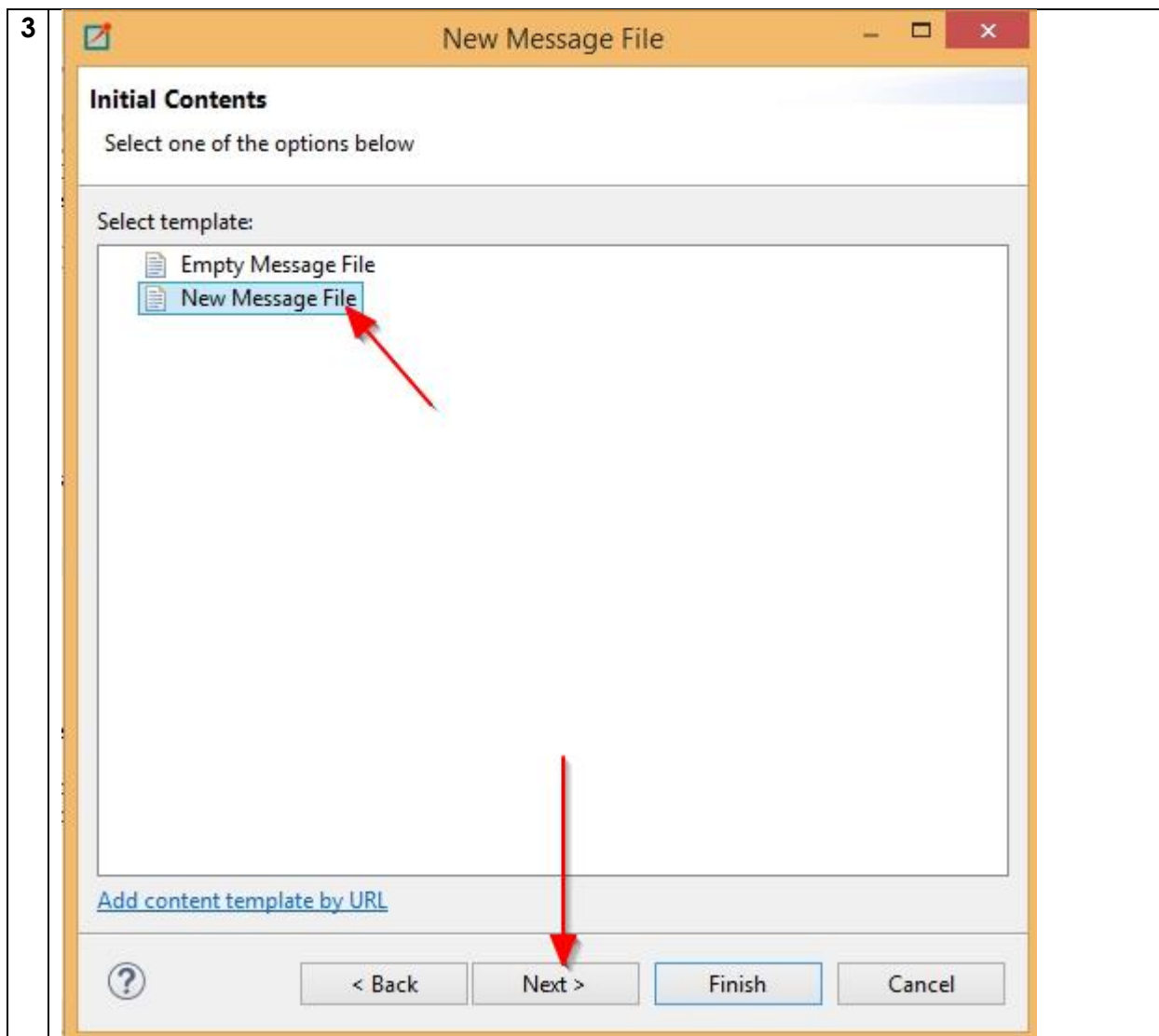
Step 1:

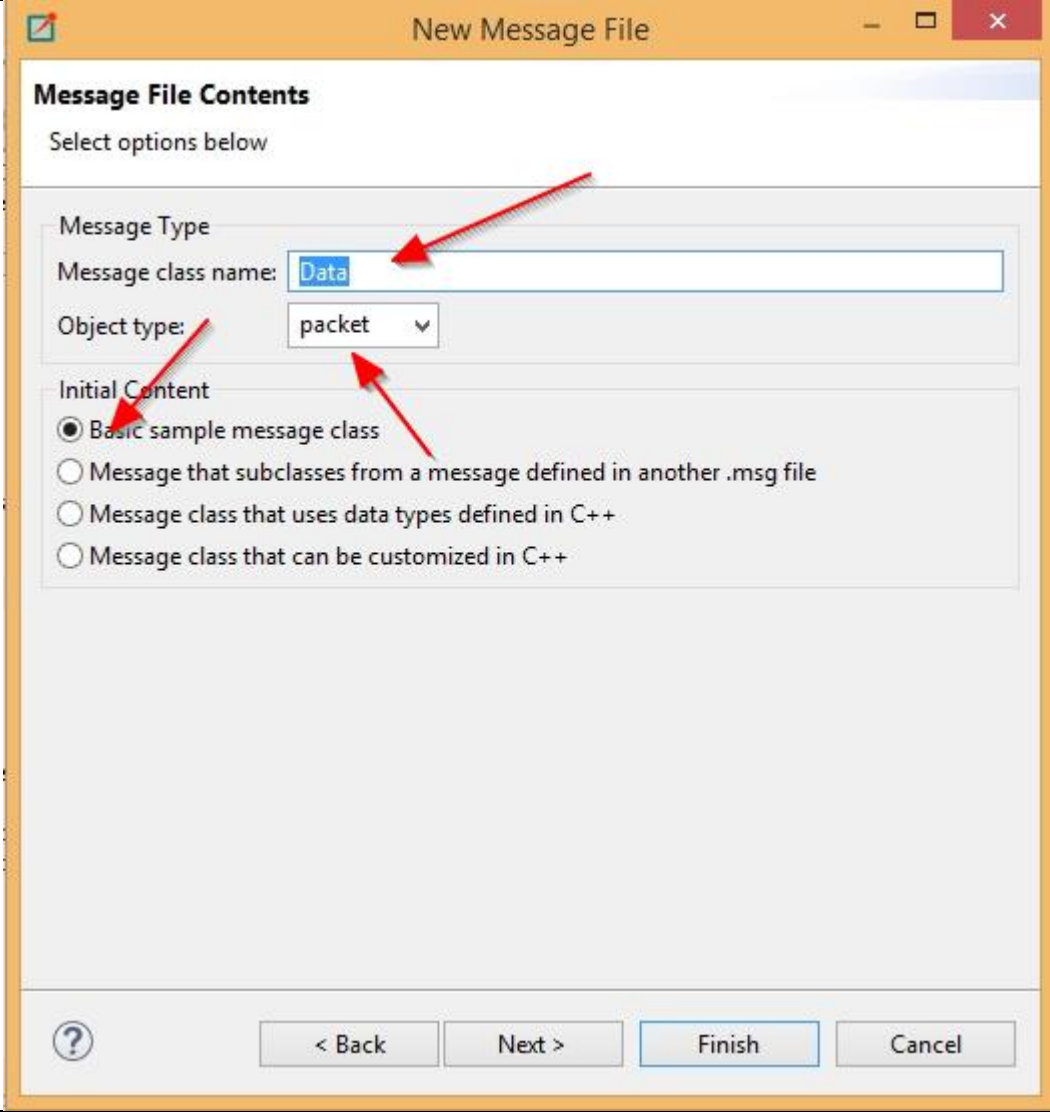
First we need to create a data packet (as data with size).

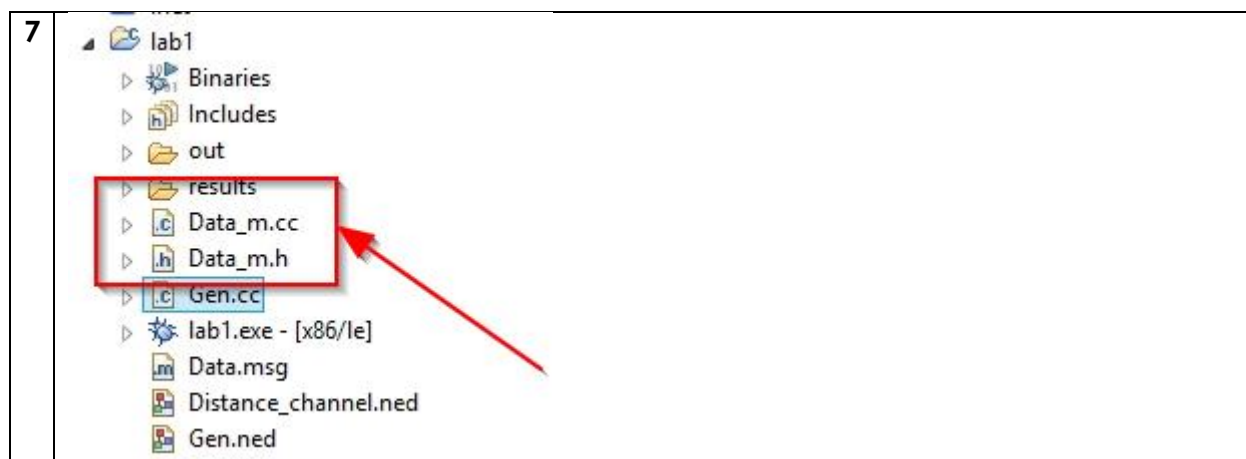
Right click on the project folder and add a new message definition and name Data.msg



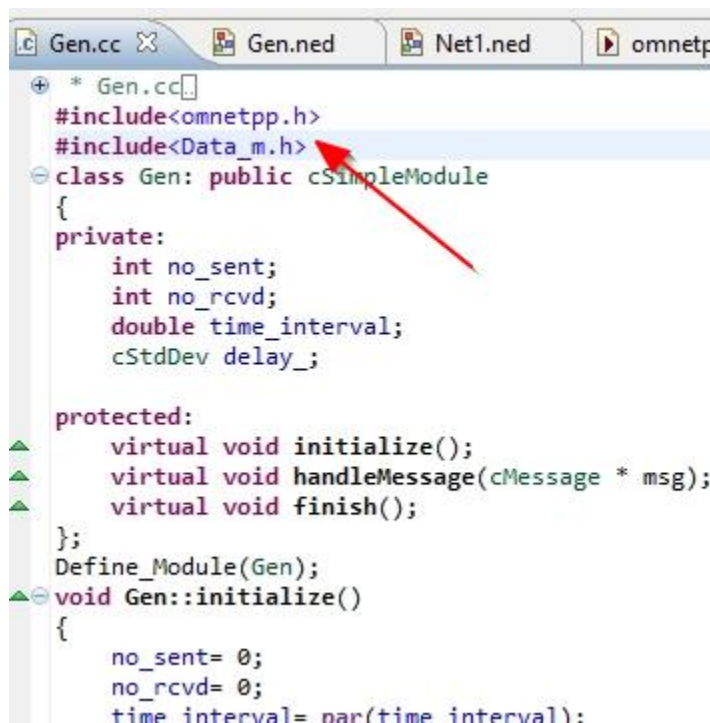




4	
5	<p>Change the Data packet fields to:</p> <pre data-bbox="256 1339 690 1808">// along with this program. // // // TODO generated message c // packet Data { int senderID; }</pre>
6	<p>Build the project once again (Ctrl+b) and observe the auto generated files (Data_m.h and Data_m.cc)</p>



To use our own packet definition add the `#include<Data_m.h>` to the Gen.cc file



The next step is to use the Data packet in the Gen class.

In this step we change the type of the message to be sent from `cMessage` to `Data` packet and add some information to the data packet (sender id, creation time and size byte length)

In the Gen class header add the following parameters:

- 1- Import the Data packet header as it will be used in the class (C++)
- 2- Add definition for GenId (as a parameter to identify each generator)
- 3- Add definition for statistic vector to calculate the delay per received data packet

```
#include<omnetpp.h>
#include<Data_m.h>
class Gen: public cSimpleModule
{
private:
    int no_sent;
    int no_rcvd;
    int GenId;
    double time_interval;
    cStdDev delay_;

protected:
    virtual void initialize();
    virtual void handleMessage(cMessage * msg);
    virtual void finish();
};
Define Module(Gen);
```

In the initiliaze function change the initialization of time_interval, GenID as a call from the ned file.

Initialize the time interval as out side variable (from ini file)

Initialize the GenId (node address) later can be used to send a packet for specific destination

```
void Gen::initialize()
{
    no_sent= 0;
    no_rcvd= 0;
    time_interval= par("time_interval");
    GenId = par("GenID");
    cMessage *msg= new cMessage();
    scheduleAt(0.01,msg);
}
```

In the handleMessage function change the code as follow:

Before send a data out:

Change the message to be sent to Data packet and add senderID and creation time

When receive a data

Cast it to Data packet

Announce from where we get it

Calculate delay and add statistic to the delay vector

```
void Gen::handleMessage(cMessage * msg)
{
    if(msg->isSelfMessage())
    {
        Data * out_msg= new Data();
        out_msg->setTimestamp(simTime());
        out_msg->setSenderID(GenId);
        send(out_msg, "out");
        no_sent++;
        scheduleAt(simTime()+time_interval,msg);
    }
    else
    {
        Data *rcvd_msg = check_and_cast<Data *>(msg);
        EV<<" a Data received from Gen with ID: "<<rcvd_msg->getSenderID()<<endl;
        delay_.collect(simTime()-rcvd_msg->getTimestamp());
        no_rcvd++;
        delete(msg);
    }
}
```

Collecting the delay results

In the finish function add a new scalar for the averaged delay:

The next step is to add the new parameters to the Gen.ned file

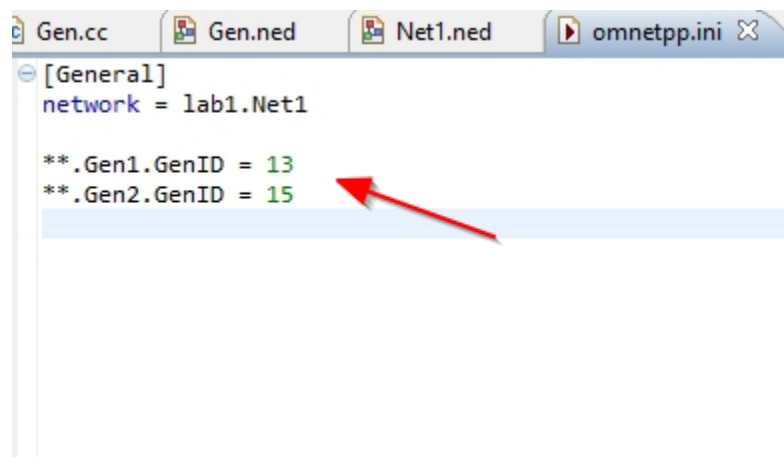
Add the following parameters:

```
//
simple Gen
{
    parameters:
        double time_interval = default(0.5);
        int GenID;

    gates:
        input in;
        output out;
}
```

Now before we run our simulation we should assign values for the GenID for each generator

In the ini file:



```

[General]
network = lab1.Net1

**.Gen1.GenID = 13
**.Gen2.GenID = 15
  
```

Run the simulation as before for 200 seconds and observe results

Browse Data

Here you can see all data that come from the files specified in the Inputs page.

All (6 / 6) Vectors (0 / 0) Scalars (6 / 6) Histograms (0 / 0)

runID filter module filter statistic name filter

Folder	File name	Config n...	R	Run id	Module	Name	Value
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen1	NUmberof received ...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen1	Number of sent mess...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen1	Average delay	0.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen2	NUmberof received ...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen2	Number of sent mess...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen2	Average delay	0.0

Delay is KOSONG why?

Lets add some delay

In the Net1.ned file add a delay to connections as:

```

package lab1;

//import Distance_channel;
//
// TODO auto-generated type
//
network Net1
{
    submodules:
        Gen1: Gen {
            @display("p=92,108");
        }
        Gen2: Gen {
            @display("p=259,108");
        }
        // link:Distance_channel; //: link;
    connections:
        Gen1.out --> {delay = 2us; } --> Gen2.in;
        Gen2.out --> {delay = 2us; } --> Gen1.in;
}

```

Now run the simulation and collect results

Browse Data

Here you can see all data that come from the files specified in the Inputs page.

All (6 / 6) Vectors (0 / 0) Scalars (6 / 6) Histograms (0 / 0)							
runID filter				module filter			statistic name filter
Folder	File name	Config n...	R	Run id	Module	Name	Value
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen1	NUmberof received ...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen1	Number of sent mess...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen1	Average delay	2.0E-6
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen2	NUmberof received ...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen2	Number of sent mess...	500.0
/lab1/r...	General-0.sca	General	0	General-0-201...	Net1.Gen2	Average delay	2.0E-6

Yes we have a fixed delay...

Can it be a random

In the connection section of the net1.ned

```
//  
  
package lab1;  
  
//import Distance_channel;  
//  
// TODO auto-generated type  
//  
network Net1  
{  
    submodules:  
        Gen1: Gen {  
            @display("p=92,108");  
        }  
        Gen2: Gen {  
            @display("p=259,108");  
        }  
        // link:Distance_channel;//: link;  
    connections:  
        Gen1.out --> {delay = uniform(4us,6us); } --> Gen2.in;  
        Gen2.out --> {delay = uniform(4us,6us); } --> Gen1.in;  
}
```