**Lab 4 Understanding tracing system in ns3**

The ns-3 tracing system is built on the concepts of independent **tracing sources** and **tracing sinks**, along with a uniform mechanism for connecting sources to sinks.

Trace sources are entities that can signal events that happen in a simulation and provide access to interesting underlying data. Trace sources are not useful by themselves; they must be connected to other pieces of code that actually do something useful with the information provided by the source. The entities that consume trace information are called trace sinks.

1. **A Simple Example**

Create myfourth.cc and add the following codes.

#include "ns3/object.h"

#include "ns3/uinteger.h"

#include "ns3/traced-value.h"

#include "ns3/trace-source-accessor.h"

#include <iostream>

using namespace ns3;

int myIndex=0; //a global index variable

class MyObject : public Object

{

public:

static TypeId GetTypeId (void)

{

static TypeId tid = TypeId ("MyObject")

.SetParent (Object::GetTypeId ())

.SetGroupName ("MyGroup")

.AddConstructor<MyObject> ()

.AddTraceSource ("MyInteger",

"An integer value to trace.",

MakeTraceSourceAccessor (&MyObject::m\_myInt),

"ns3::TracedValueCallback::Int32")

;

return tid;

}

MyObject () {};

TracedValue<int32\_t> m\_myInt;

};

The AddTraceSource provides the “hooks” used for connecting the trace source to the outside world through the Config system. The first argument is a name for this trace source, which makes it visible in the Config system. The second argument is a help string. Now look at the third argument, in fact focus on the argument of the third argument: &MyObject::m\_myInt. This is the TracedValue which is being added to the class; it is always a class data member. The final argument is the name of a typedef for the TracedValue type, as a string.

The TracedValue< > declaration provides the infrastructure that drives the callback process. Any time the underlying value is changed the TracedValue mechanism will provide both the old and the new value of that variable, in this case an int32\_t value. **So far we have created a tracing source.**

**The next function IntTrace ( ) is a tracing sink.**

**void** IntTrace (**int32\_t** oldValue, **int32\_t** newValue)

{

std::cout << "Iteration:" << myIndex << "\t" << "Oldvalue:"<< oldValue << "\t"<<"Newvalue:"<< newValue << std::endl;

}

We will connect these two in the main program as follows.

**int**

main (**int** argc, **char** \*argv[])

{

Ptr<MyObject> myObject = CreateObject<MyObject> ();

myObject->TraceConnectWithoutContext ("MyInteger", MakeCallback(&IntTrace));

for (myIndex=1; myIndex<10; myIndex++)

{

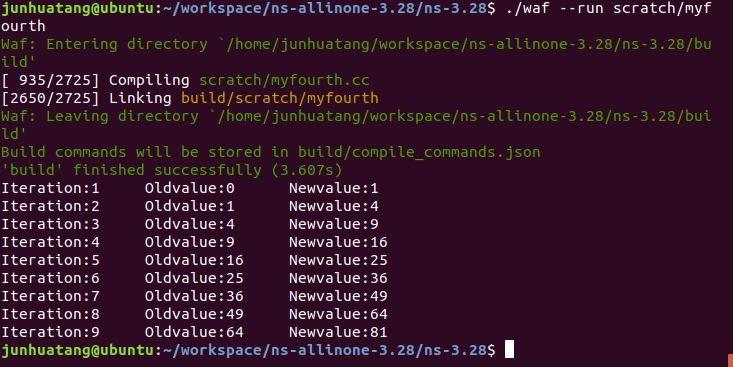
myObject->m\_myInt = myIndex\* myIndex;

}

}

$./waf

$./waf --run scratch/myfourth



1. **Connect one more trace sink to the “MyInteger” trace source**

Include one more header file in myfourth.cc：

#include "ns3/internet-module.h" // for OutputStreamWrapper

Add the following function (a trace sink) after the IntTrace ( ) function:

static void

IntRecord (Ptr<OutputStreamWrapper> stream, int32\_t oldValue, int32\_t newValue)

{

\*stream->GetStream () << myIndex << "\t" << newValue << std::endl;

}

In the main () program, connect the trace source “MyInteger” with the trace sink “IntRecord” using the following codes

std::string filename = "myfourth.dat";

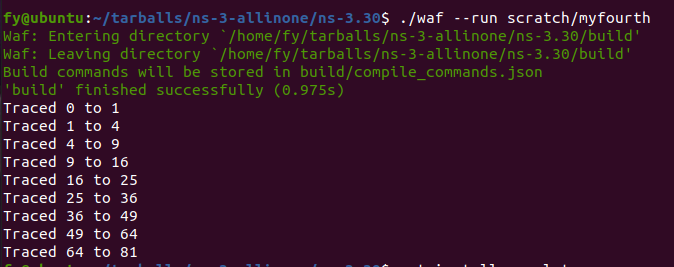
std::ios::openmode filemode = std::ios::out;

Ptr<OutputStreamWrapper> stream = Create<OutputStreamWrapper> (filename, filemode);

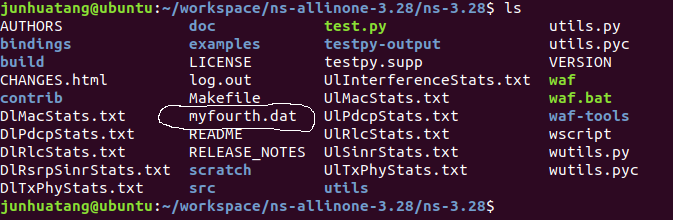
myObject->TraceConnectWithoutContext ("MyInteger", MakeBoundCallback (&IntRecord, stream));

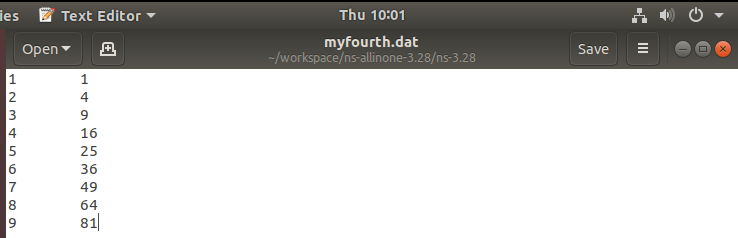
$./waf

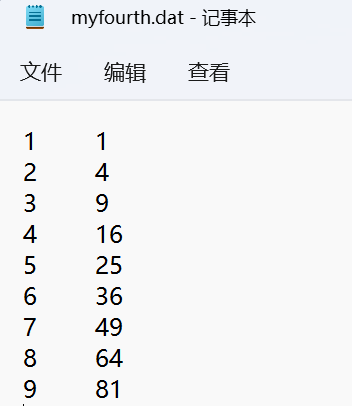
$./waf --run scratch/myfourth



$ls (make sure myfourth.dat is there)







Use gnuplot to draw the graph. (May need to install gnuplot package if you have not done so.)

$ gnuplot

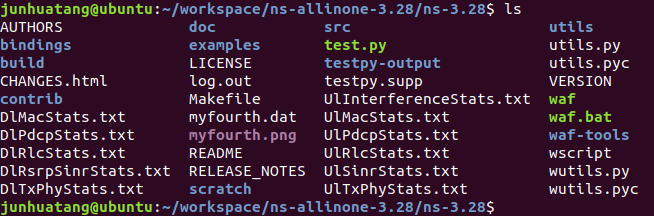
gnuplot> set terminal png size 640,480

gnuplot> set output "myfourth.png"

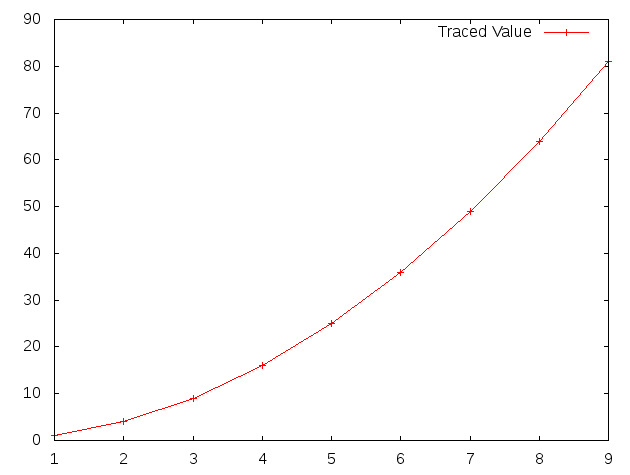
gnuplot> plot "myfourth.dat" using 1:2 title 'Traced Value' with linespoints

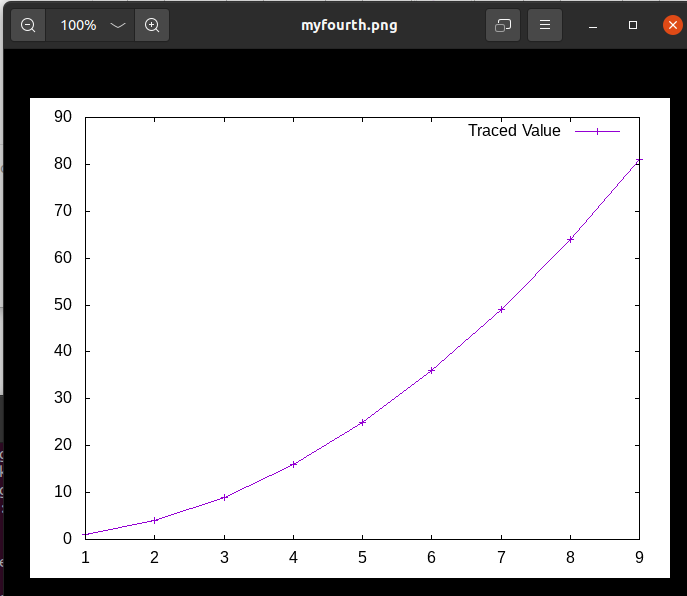
gnuplot> exit

$ls (make sure myfourth.png is created.)



Open lab3myfourth.png and it should look like the following.



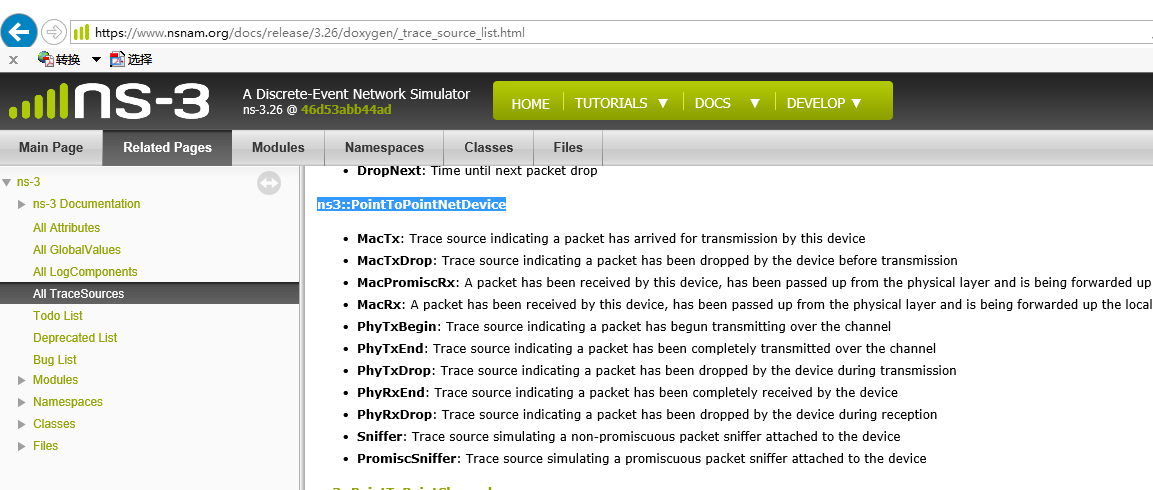


1. **Using Config to connect to trace sources**

In this part of the tutorial, we are going to use myfirst.cc as an example. So go ahead and open myfirst.cc.

**3.1 find a tracing source**

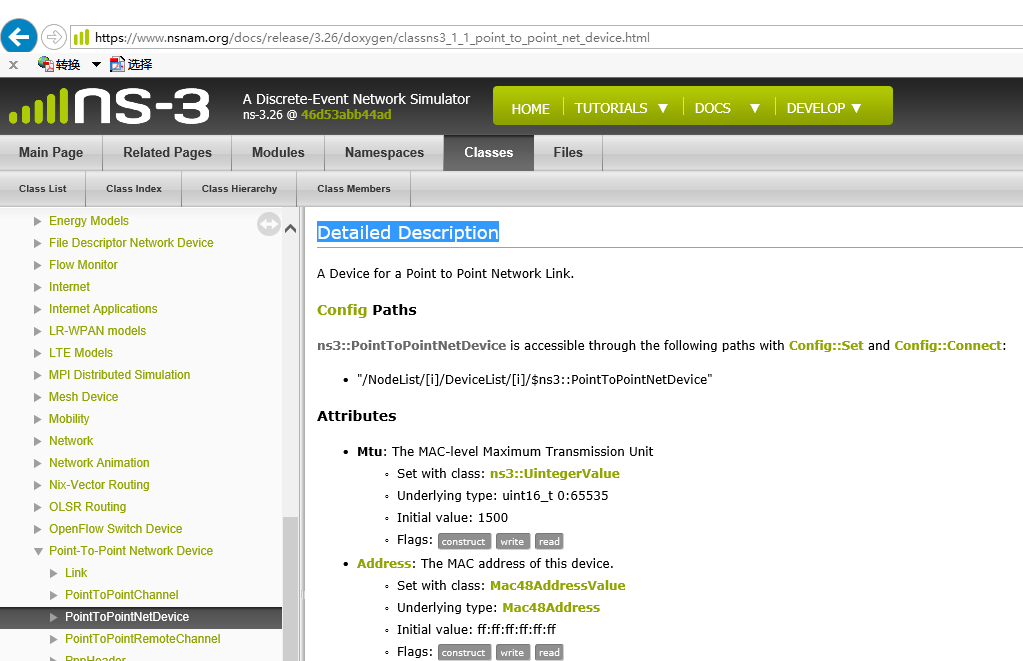
Go to <https://www.nsnam.org/docs/release/3.28/doxygen/_trace_source_list.html> and find ns3::PointToPointNetDevice

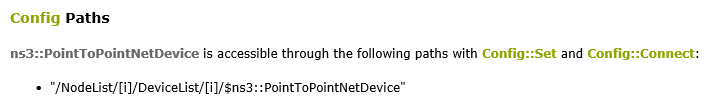


We will try to write a TraceSink and connect to MacTx.

**3.2 Config paths of a tracing source**

Click ns3::PointToPointNetDevice in the figure above, on the page of ns3::PointToPointNetDevice class shown below, scroll down to “Detailed Description”.





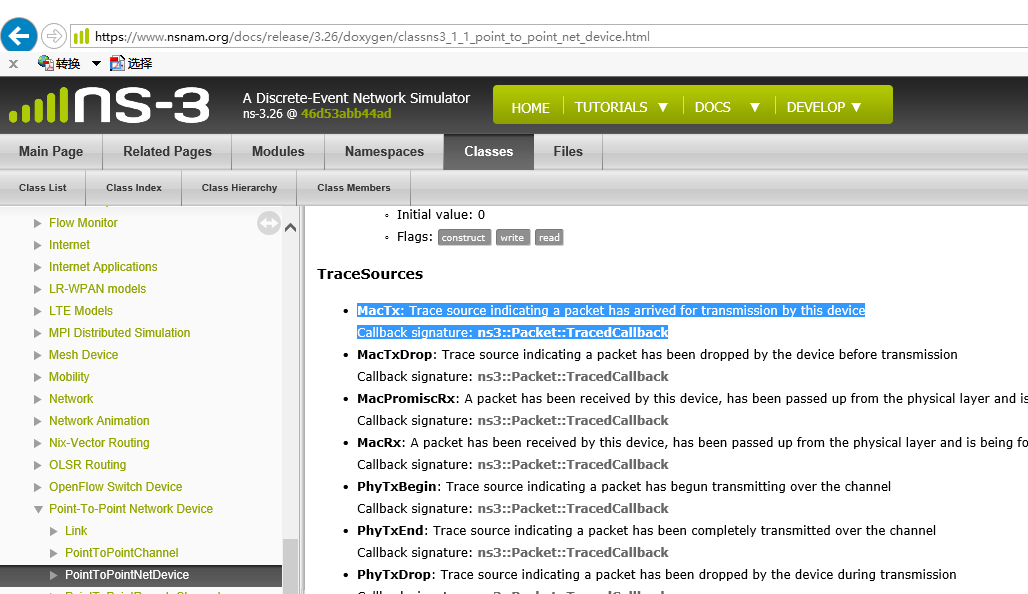
e.g. in myfirst.cc

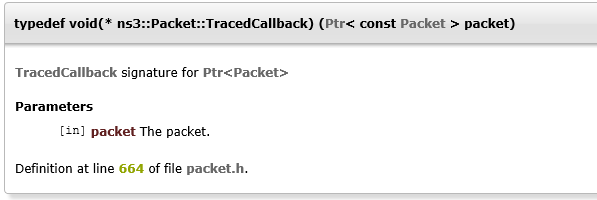
/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacTx

refers to the TraceSource MacTx on Node 1 Device 0.

* 1. **Callback signature:** the return type and formal arguments of callback function

At the end of “Detailed Description”, click callback signature: ns3::Packet::TracedCallback





It shows that your callback function has one argument: (Ptr< const Packet > packet), you can check ns3::Packet Class Reference at <https://www.nsnam.org/docs/release/3.28/doxygen/classns3_1_1_packet.html> .

Thus a very simple trace sink function may look like:

static void

MyPacketTrace (Ptr< const Packet > packet)

{

std::cout << "My traced packet, size: " << packet->GetSize() << std::endl;

}

Put the above codes in myfirst.cc just after NS\_LOG\_COMPONENT\_DEFINE ("FirstScriptExample");

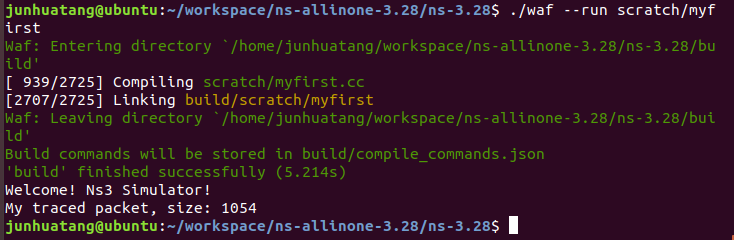
* 1. **Connect trace sink to trace source (Config::ConnectWithoutContext())**

In the main() program, just before Simulator::Run (); add the following line

Config::ConnectWithoutContext("/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacTx", MakeCallback(&MyPacketTrace));

$./waf

$./waf --run scratch/myfirst



* 1. Connect with: Config::Connect()

Config::Connect("/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacTx", MakeCallback(&MyPacketTraceWithContext));

Config::Connect will put the context(config path) string as the first parameter to the call back function, therefore your callback function in this example will have two arguments (std::string context, Ptr< const Packet > packet).

Add the following trace sink to myfirst.cc

static void

MyPacketTraceWithContext (std::string context, Ptr< const Packet > packet)

{

std::cout << context << "My traced packet, size: " << packet->GetSize() << std::endl;

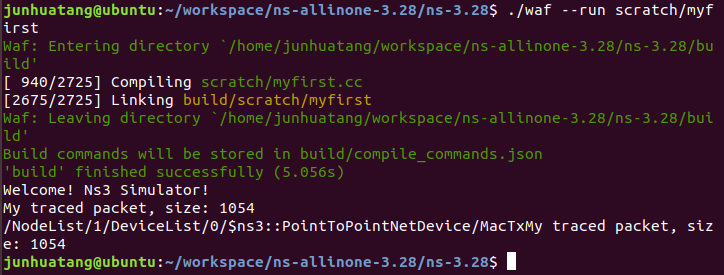
}

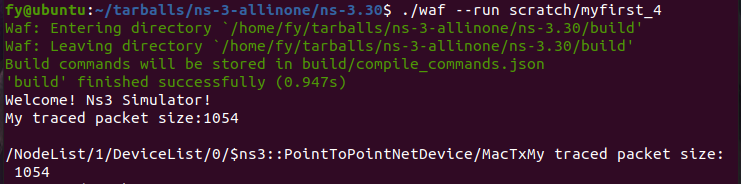
And add the following line to connect the trace sink to the trace source of "/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacTx"

Config::Connect("/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/MacTx", MakeCallback(&MyPacketTraceWithContext));

$./waf

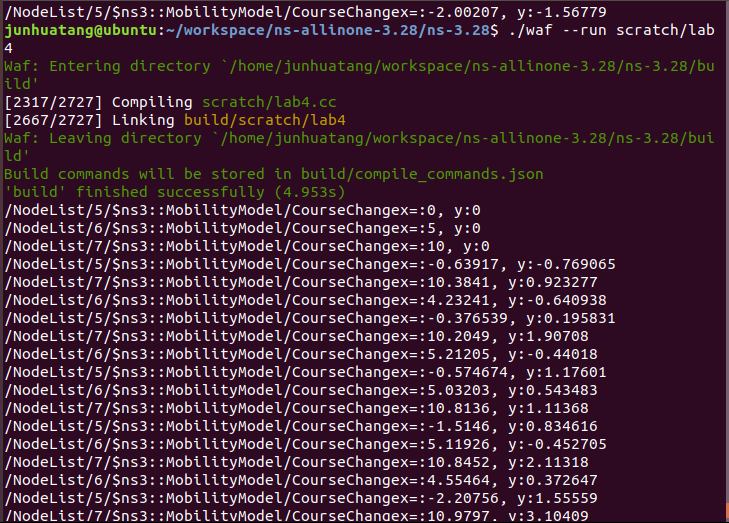
$./waf --run scratch/myfirst

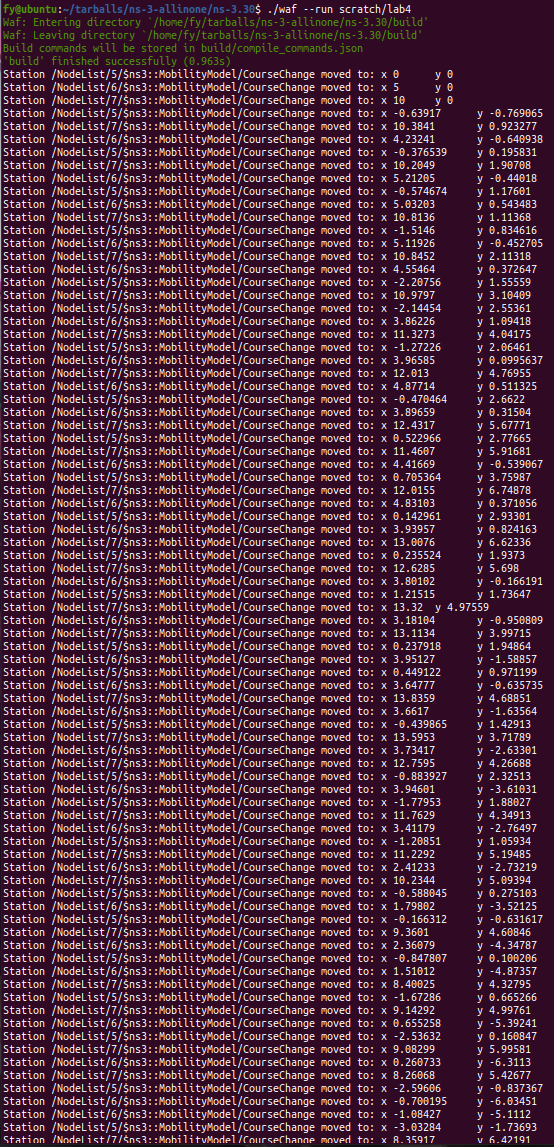




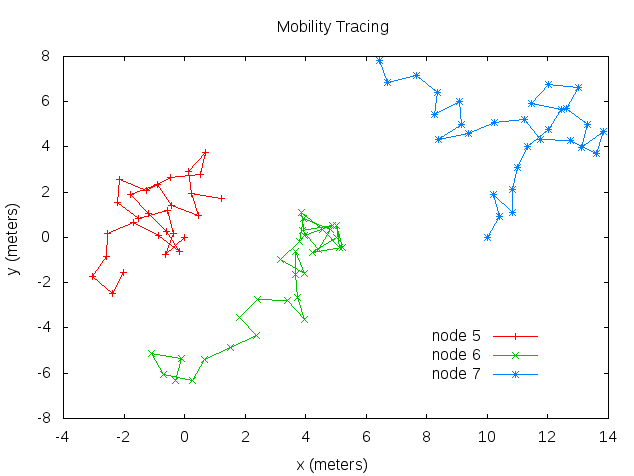
**Exercise**: Trace the mobility of wifi station nodes in examples/tutorial/third.cc, name your source code lab4.cc.

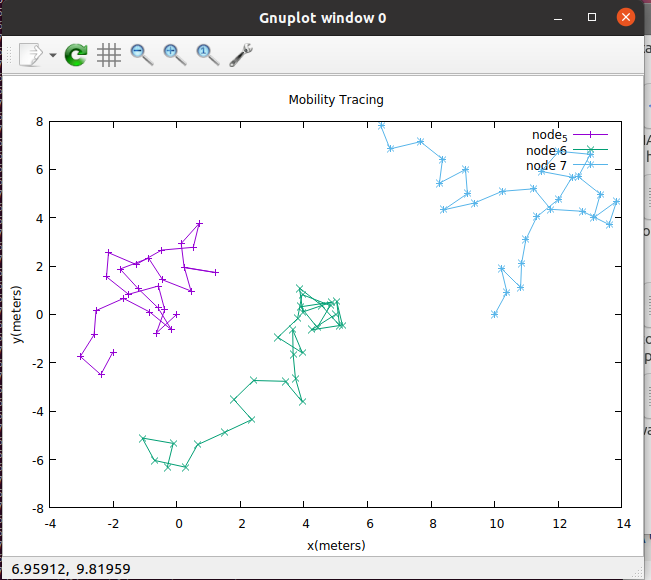
1. Print out the position (x,y) of wifi station nodes as they move





2. Use gnuplot to draw the traces of wifi station nodes

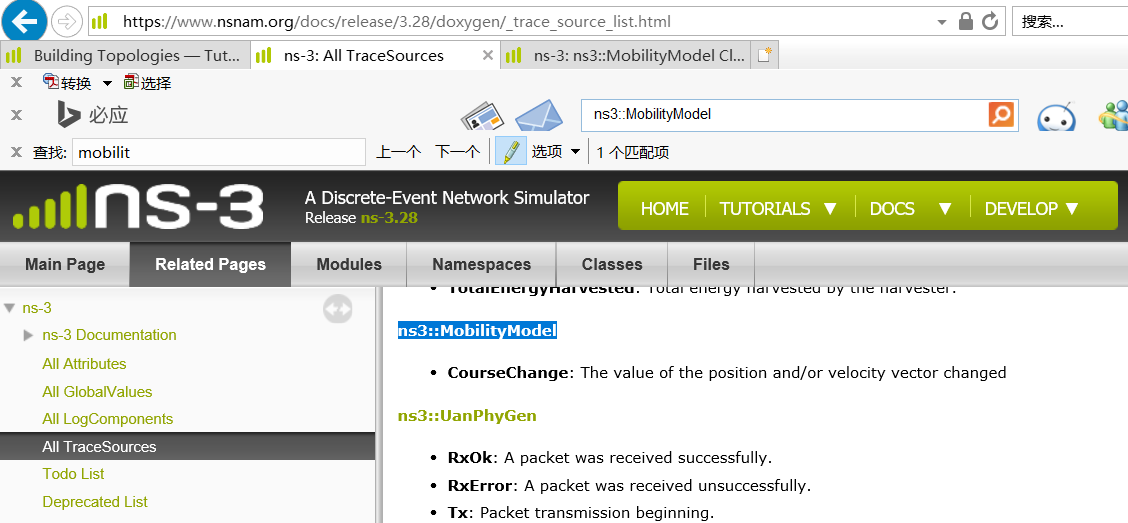




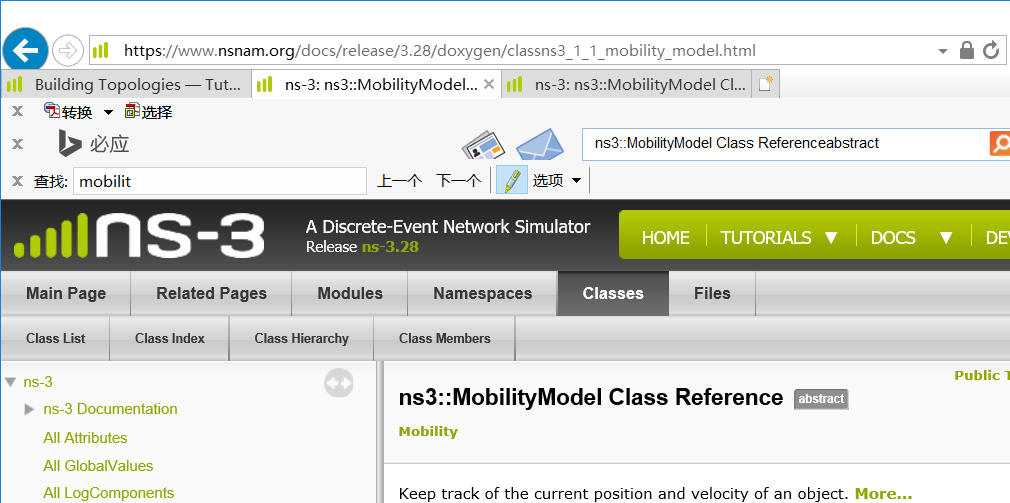
Hints:

1. Use trace source of ns3::MobilityModel/CourseChange

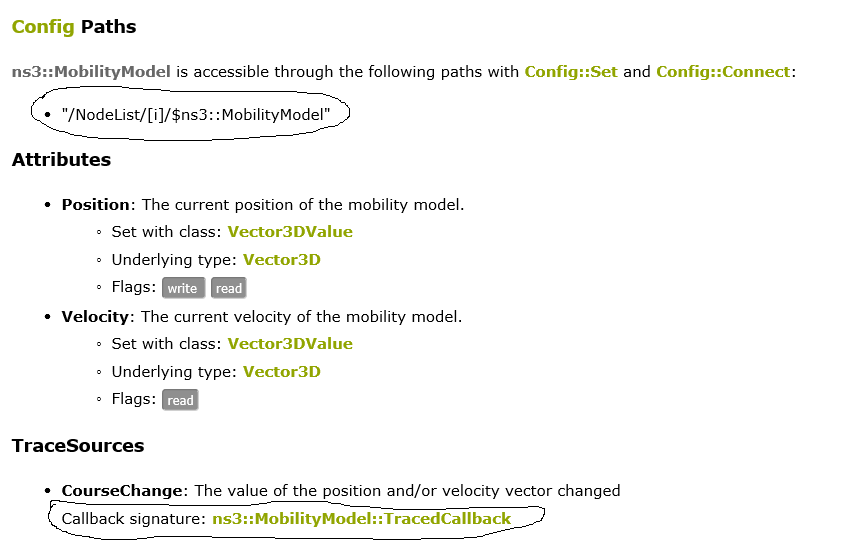
First find **ns3::MobilityModel** on https://www.nsnam.org/docs/release/3.28/doxygen/\_trace\_source\_list.html



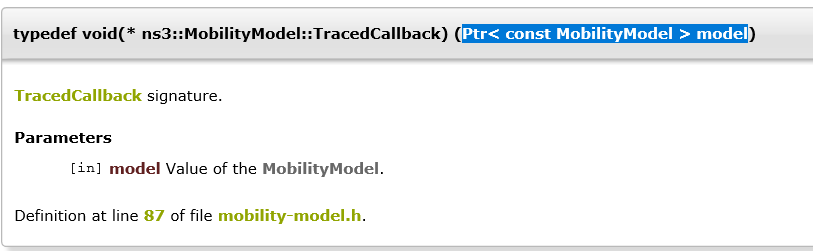
Then click on **ns3::MobilityModel** and go to **ns3::MobilityModel Class Reference** site



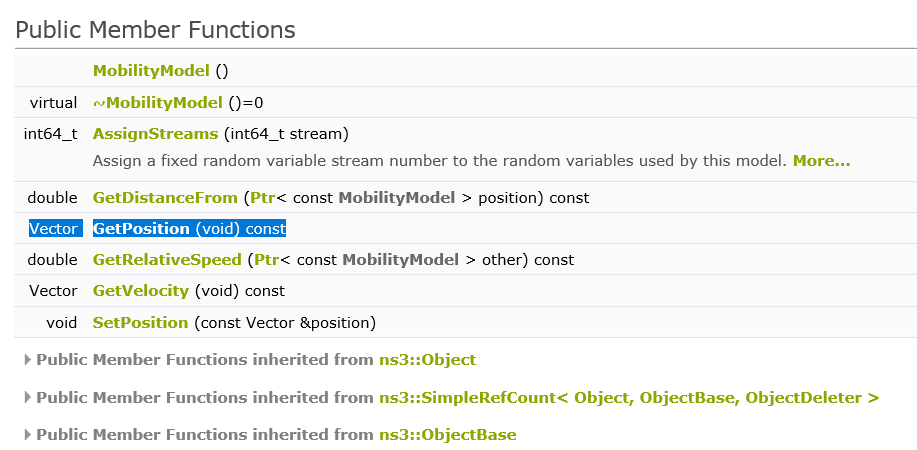
Scroll down the page to locate the **Config Paths** and **TraceSources**



Click on Callback signature: [ns3::MobilityModel::TracedCallback](https://www.nsnam.org/docs/release/3.28/doxygen/classns3_1_1_mobility_model.html#a1bcfc7397046b2b04713a52884a417af) to find the return value and arguments of the callback function.

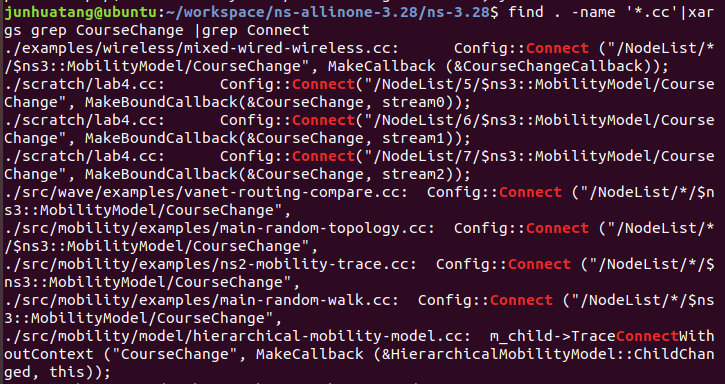


Continue to click **Mobility Model** to find the public functions of the **MobilityModel** class



1. Find example codes:

$find . -name '\*.cc'|xargs grep CourseChange |grep Connect



1. For more information on gnuplot:

http://people.duke.edu/~hpgavin/gnuplot.html

Turn in:

1. lab4.cc
2. lab4-report
   1. with screenshots of simulation results
   2. any thoughts on the exercises