Cairo University
Faculty of Engineering
Computer Department

Computer Networks-1
CMP405-A /CMPN405
Fall 2023

Using Parameters in omnet++

lab2

Today's Outline

- Using Parameters in omnet++
- Random number generators
- Lab2 Requirement
- How to import projects in omnet++

Parameters in NED language

- Parameters are variables that belong to a module.
- Simple and compound modules can have parameters as well.

```
simple Queue
{
    parameters:
        int capacity;
        @class(mylib::Queue);
        @display("i=block/queue");
    gates:
        input in;
        output out;
}
```

This is a .NED file

Parameters in NED language

- Simple modules can be extended (or specialized) via subclassing using parameters

```
simple Queue
{
    int capacity;
    ...
}

simple BoundedQueue extends Queue
{
    capacity = 10;
}
```

Parameters in NED language

- Parameters can be of type double, int ,bool , string and other custom types(units).

```
simple App
{
    parameters:
        string protocol;        // protocol to use: "UDP" / "IP" / "ICMP" / ...
        int destAddress;        // destination address
        volatile double sendInterval @unit(s) = default(exponential(1s));
                                // time between generating packets
        volatile int packetLength @unit(byte) = default(100B);
                                // length of one packet
        volatile int timeToLive = default(32);
                                // maximum number of network hops to survive
    gates:
        input in;
        output out;
}
```

Parameters in NED language

- @unit() declaration specify the measurement unit for the parameter.
- @unit(s) accepts milliseconds, nanoseconds, minutes, hours, etc.,
- @unit(byte) accepts kilobytes, megabytes, etc. as well.

Unit	Name	Value
d	day	86400s
h	hour	3600s
min	minute	60s
s	second	
ms	millisecond	0.001s
us	microsecond	1e-6s
ns	nanosecond	1e-9s
ps	picosecond	1e-12s
fs	femtosecond	1e-15s
as	attosecond	1e-18s
bps	bit/sec	
kbps	kilobit/sec	1000bps
Mbps	megabit/sec	1e6bps
Gbps	gigabit/sec	1e9bps
Tbps	terabit/sec	1e12bps
B	byte	8b

Parameters in NED language

- Parameters can be of type double, int ,bool , string and other custom types(units).
- They can also be declared volatile

```
simple App
{
    parameters:
        string protocol;        // protocol to use: "UDP" / "IP" / "ICMP" / ...
        int destAddress;        // destination address
        volatile double sendInterval @unit(s) = default(exponential(1s));
                                // time between generating packets
        volatile int packetLength @unit(byte) = default(100B);
                                // length of one packet
        volatile int timeToLive = default(32);
                                // maximum number of network hops to survive
    gates:
        input in;
        output out;
}
```

What is the use of volatile?

Parameters in NED language

Parameters can be assigned values in different ways:

- 1) using const /(default) values
- 2) using default function output

```
parameters:
    string protocol;        // protocol to use: "UDP" / "IP" / "ICMP" / ...
    int destAddress;        // destination address
    volatile double sendInterval @unit(s) = default(exponential(1s));
                            // time between generating packets
    volatile int packetLength @unit(byte) = default(100B);
                            // length of one packet
    volatile int timeToLive = default(32);
                            // maximum number of network hops to survive
```

Parameters in NED language

- Parameters can be assigned values in different ways
- 3) in the submodule instantiation inside compound modules

```
module Host
{
    submodules:
        ping : PingApp {
            packetLength = 128B; // always ping with 128-byte packets
        }
        ...
}
```

Parameters in NED language

- Parameters can be assigned values in different ways
- 4) using wildcards in any .NED file (especially in the .ini configuration file)

```
network Network
{
    parameters:
        **.timeToLive = default(3);
        **.destAddress = default(0);
    submodules:
        host0: Host;
        host1: Host;
        ...
}
```

This is inside the network .NED file

equivalent to **.sendIinterval

```
**.ping.sendInterval = 500ms
```

This is inside the .ini file

searches for this ain all files in the project bec she has used **

Parameters in NED language

- * : matches zero or more characters except dot (.) inside the level i am in
- ** : matches zero or more characters (any character) lower level
- {a-f} : set: matches a character in the range a-f
- {^a-f}: negated set: matches a character NOT in the range a-f
- [38..150] : index range: any number in square brackets in the range 38..150, inclusive; both limits are optional

Parameters in NED language

NEXT SECTION

- Parameters can be assigned values in different ways
- 4) using wildcards in any .NED file (especially in the .ini configuration file)

```
network Network
{
    parameters:
        host[*].ping.timeToLive = default(3);
        host[0..49].ping.destAddress = default(50);
        host[50..].ping.destAddress = default(0);

    submodules:
        host[100]: Host;
        ...
}
```

This is inside the network .NED file

Parameters linkage to the .cc files

What if we want to read parameter values inside the .cc behavior files ?

Parameters linkage to the .cc files

- Using the function par

  par("parameter name").conversion value()

Example:

Inside the Node.ned file

```
//
simple Node
{
    parameters:
    int limit=default(5);
    gates:
    input in;
    output out;
}
```

Inside the Node.cc file

```
void Node::handleMessage(cMessage *msg)
{
    // TODO - Generated method body
    if (counter == par("limit").intValue())
    {
        EV << "reaching the limit and deleting the message"<<endl;
         cancelAndDelete(msg);
    }
```

if(counter==limit) --> don't understand the limit var

Parameters linkage to the .cc files

- Other conversion type functions:

boolValue(), longValue(), doubleValue(), stringValue(), stdstringValue().

- There are also overloaded type cast operators for the corresponding types(bool; int, long, double)

```
long numJobs = par("numJobs").longValue();
double processingDelay = par("processingDelay"); // using operator double()
```

Use first way better

Parameters linkage to the .cc files

- Careful not to confuse the .NED parameters with the .cc class data members.
- Inside the Node.h file

```
 */
class Node : public cSimpleModule
{
  protected:
    int counter;
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

#endif
```

Inside the Node.cc file

```
void Node::handleMessage(cMessage *msg)
{
    // TODO - Generated method body
    if (counter == par("limit").intValue())
    {
        EV << "reaching the limit and deleting the message"<<endl;
         cancelAndDelete(msg);
    }
```

Random number generators

In NED Language

- Omnet++ offers utility functions to sample from many different distributions.
- For example: the uniform function returns a random number between a and b.

uniform(int a, int b) [a,b)

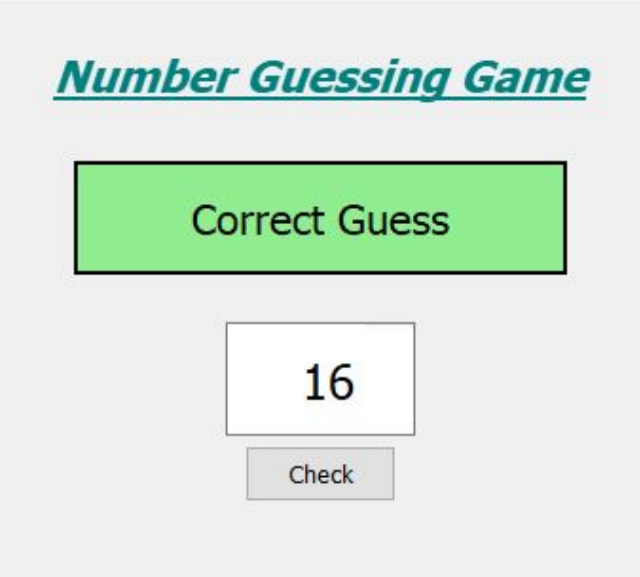
- To generate a random start every time, set the seed to any number in the .ini file like this:

```
seed-set=522
```

Omnet++
Lab2 requirement

- Implement a random guessing game
- One of the nodes acts as player1 and the other as player2.

This is not how your omnet GUI should look like , it is just for illustration.



Omnet++
Lab2 requirement

- Player1 generates a random number at the beginning of the game and prints it to the console. “note that the number should be constant through the game afterwards.
- The number ranges from 0 to 10 .
- Player1 signals player 2 to send messages containing his guesses one at a time.

Omnet++
Lab2 requirement

- Player2 randomly chooses a guess and sends it to player1 .
- The guess ranges from 0 to 10 .
- If the guess is not correct, player1 sends to player2 a “`wrong guess`” message .
- If the guess is correct, player1 sends a “`correct guess`” message to player2 and the simulation is finished.

Omnet++
Lab2 requirement

- Print all the messages to the console also.
- Print the guessing number at the beginning .
- Hint: you may need to implement two different simple modules, one for player1 and one for player2.
- Hint: there is a msg->setName() function that changes the name property.
- Don’t send the real number to player2 clearly :D
- *Free your dynamically memory allocations carefully.*

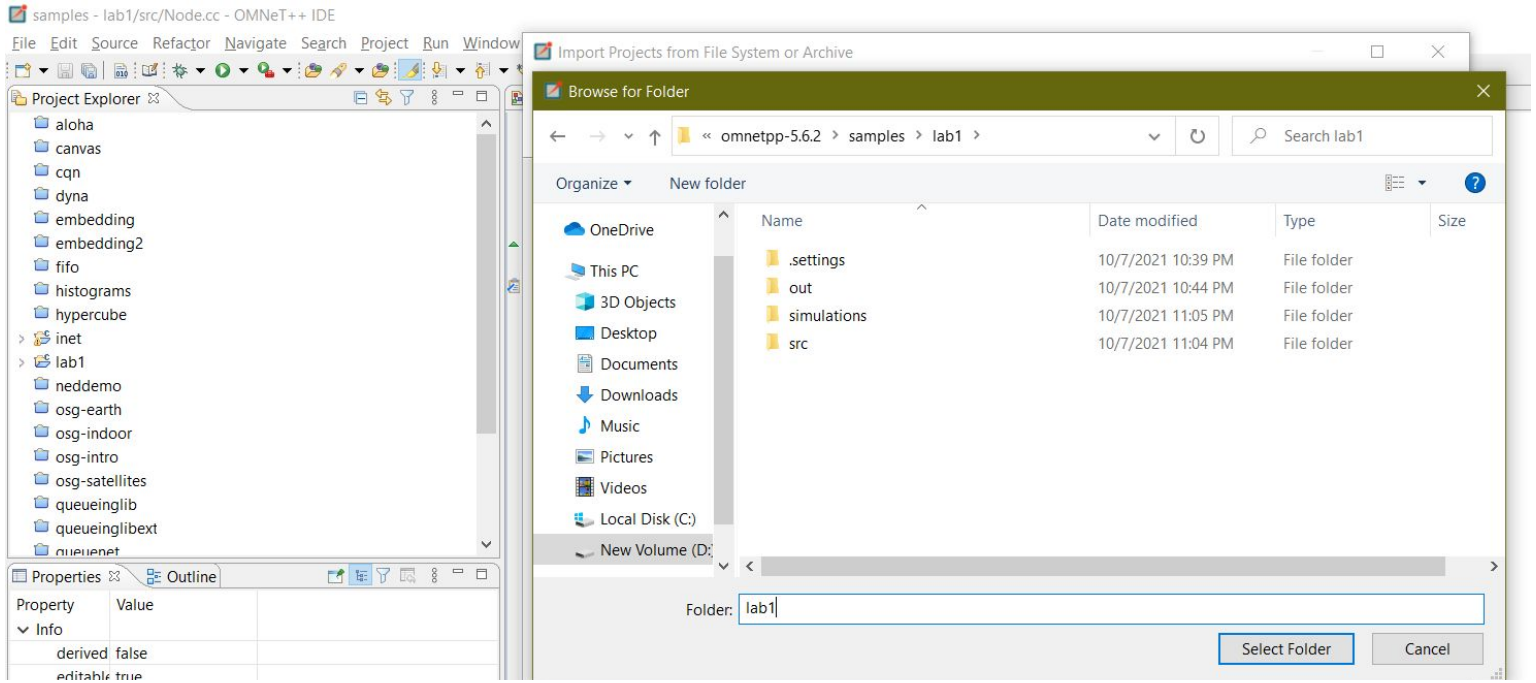
send(msg)
cancelanddelete(msg)
unexpected behaviour according did you delete it first or he has read t first

Omnet++
Lab1 delivery

- How to import a project in omnet++ ?
- 1) inside omnet++ go to file -> open project from file system.
- 2) write in the import source the exact directory to your unzipped project file.

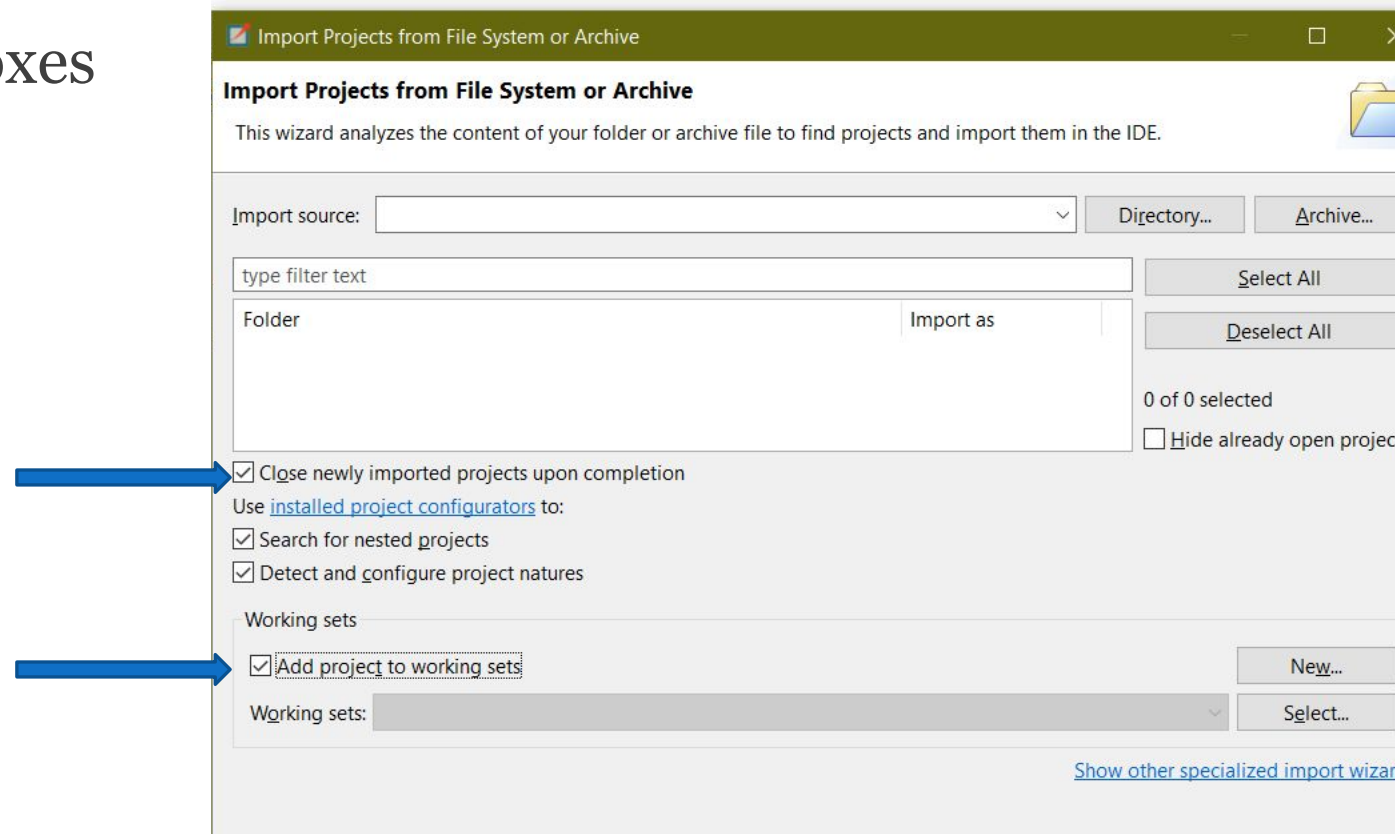
Omnet++
Lab1 delivery

- 3) make sure that the directory path you provide has the src folder inside it directory.



Omnet++
Lab1 delivery

- *How to import a project in omnet++ ?*
- 4) check these checkboxes and finish.



Omnet++ helper functions

- `atoi(cstr)` to convert a c_str, that contains a number, to int.
- `int(uniform(0,10))` to generate random integers from 0 to 9.
- `seed-set=any number` to make the generator random.
- `par("parname").intValue()` to convert the parameter to int.

Thank You !!