PART I: Programming

Writing a Nano-World Virtual Experiment

Download the actual vexp release from http://sourceforge.net/projects/vexp

The Demo Virtual Experiment

The only thing you have to do, is to write a class which extends the class nano.compute.Simulator and to implement the following methods:
The constructor.
The run() method.

The Constructor

It is common to make use of the constructor of the superclass and add some specific statements. For this purpose you may use an init() method. So your constructor becomes the following form:

```
public DemoSimulator(){
  super();
  init();
}
```

We will look at the init() method later.

The run() method

The simulation itself, i. e. the computation of physical quantities takes place in the run method. It typically consists of a while loop. Since the simulator runs in an independent thread, you have to let this thread sleep from time to time. So your computer gets time to do other computations than just simulating a physical system.

```
public void run(){
  while(true){
     … your simulation code
     System.out.print("I'm simulating.");
     try{sleep(100);}catch (InterruptedException e){}  //Sleep
  }
}
```

Setting the sleep-time

In the run method we added the line

```
try{sleep(100);}catch (InterruptedException e){}
```

so the simulation thread will sleep for 100 milliseconds and the computer processor may take care of other processes. If there is nothing else to do, it is possible, that your thread continues before the 100 milliseconds are over.
When the thread is interrupted from outside, an InterruptedException is thrown, which you need to catch.
Use sleep times corresponding to your system not smaller than 1 to 5 ms.

Run() method of the DemoSimulator

```
10  public void run(){
11     byte[] data = new byte[ss];
12     double tmp;
13     while(true){
14        while(running){
15         for(int yi=0;yi<ss;yi++){ // y direction
16            for(int xi=0;xi<ss;xi++){ // x direction
17
18              //simple trigonometirc function
19              tmp = gd("a")*8*Math.sin(gd("b")*(double)xi/(double)ss*4*Math.PI+
20                                  gd("c")*(double)yi/(double)ss*Math.PI/2);
21
22              System.out.println(tmp+" a:"+gd("a")+" b:"+gd("b")+" c:"+gd("c"));
23              if ((tmp>=-128) && (tmp <=127)){data[xi]=(byte)tmp;}
24                else{
25                  if(tmp < -128){data[xi]=-128;}
26                     else{data[xi]=127;};};
27            } //
28            data[0]=(byte)(yi-ss/2); //first byte stores the line number from -128 to 128
29
30            // send Stream to Clients
31            try{source.write( data,0, 256);}catch(IOException e){}
32            try{this.sleep(400);}catch (InterruptedException e){}
33
34          }
35        } //END RUNNING
36       }  //END WHILE(TRUE)
37     }    //END RUN
```

Line 11,12:	Definition of local variables for the simulation
Line 13	Calculation Loop
Line 14	Start and Stop control for the DemoSimulation
Line 15,16	x,y raster from 0 to 255
Line 19	Simulation Formula
Line 23-26	wirte the calculated data in the data array, if the result is between –128 and 127 (Byte range). Results smaller than –128 are set to –128, results larger than 127 are set to 127.
Line 28	The y Position is store in the first Byte (for the nanoviewer)
Line 31	Sends the result array (256 Bytes)
Line 32	Sleeps the thread for a while (Important the thread should always sleep direct after transmitting the results.

The start() command

Now, your simulation is ready to go. You just have to start it using the start() command, This can be done from the Controller (The creator of your Simulation) or directly in the constructor:

```
public DemoSimulator(){
  super();
  init();
}
```

Parameters

Parameters which should be controllable from outside have to be declared as "Labelled Variables". There exist 4 types of "Labelled Variables": double, int, String, boolean.

Adding Parameters

In the init() method, declare your parameters by type, name and initial value:

```
Simulation: r = a * sin (b*x+c*y)

public void init(){
        addDouble("a",3.0);
        addDouble("b",5.0);
        addDouble("c",0.6);


    }
```

Adding Commands

In the init() method you may declare your commands and the corresponding CommandExecutor:

```
public void init(){
        addDouble("a",3.0);
        addDouble("b",5.0);
        addDouble("c",0.6);


        addCommand("startdemo", new startCommandExecutor());
        addCommand("stopdemo", new stopCommandExecutor());

}
```

CommandExecutors

The corresponding CommandExecutor is typically implemented as a inner class and is inherited from the abstract class nano.compute.CommandExecutor. You have to implement the abstract method execute:

```
abstract public void execute(Hashtable tags) throws ParseException;
```

The execute method does nothing else than calling a method of the Simulater class itself (not the inner class!)
For the startdemo example, it reads like this:

```
class startCommandExecutor extends CommandExecutor{
        public void execute(Hashtable tags){
            start_simulation();
        }
    }
```

Accessing the "Labelled Variables"

In order to access the Labelled Variables you defined for example with the addDouble command, you need to use the setters and getters:

Type	Getter	Shortcut	Setter	Shortcut
Double	double getDouble(String)	gd(String)	Void setDouble(String, double)	Sd(String,double)
Integer	getInt			
Boolean	getBoolean			
String	getString			

Example DemoSimulator

```
//simple trigonometirc function
        tmp = gd("a")*8*Math.sin(gd("b")*(double)xi/(double)ss*4*Math.PI+
                gd("c")*(double)yi/(double)ss*Math.PI/2);
```

The parameters a,b,c are read by the method gd("name")

PART II: Compiling

Compiling under Windows NT or 2000:

Unzip vexp.zip
cd vexp
ant clean
ant

or:

Unzip vexp.zip
cd vexp
compile.bat client
compile.bat server

Compiling under Linux, Unix or os x:

Unzip vexp.zip
cd vexp
ant clean
ant

or:

Unzip vexp.zip
cd vexp
./compile.sh client
./compile.sh server

PART III: Configuration

Some configuration is needed to run your DemoSimulator:

- Define which virtual experiment will be used.
- Define the global parameters
- Adapt the experiment StartScript

Define which virtual experiment will be used.

conf/config.dat

The config.dat file defines the communication ports and the virtual experiment class.
To start the DemoSimulator change your config.dat

#Config file for initialisation of vexp-server.
#
#command=init_by_db roomname=testroom
command=initports eventport=5000 streamport=5002
command=initsource name=DemoSimulator

Define the global parameters

All global parameters which should be represented by a tachometer or an other kind of regulator must be defined in the meta file. Write the following 3 lines into the conf/meta file:

label=a guitype=vollkreis min=-200 max=200 Steigung=10 Name=Amplitude
label=b guitype=vollkreis min=0.5 max=10 Steigung=10 Name=Frequency
label=c guitype=vollkreis min=0.5 max=10 Steigung=10 Name=Phase

label:	gd(“a”) get the value of the label a
guitype:	define the view and features of the regulator
min:	min value
max:	max value
Steigung:	not implemented

Adapt the experiment StartScript

The server runs first the StartScript.dat
To run the DemoSimulator you have to use the following script

conf/StartScript.dat

command=start
#command=set name=springconstant value=3
#command=set name=scansize value=4
#command=approach
command=startdemo

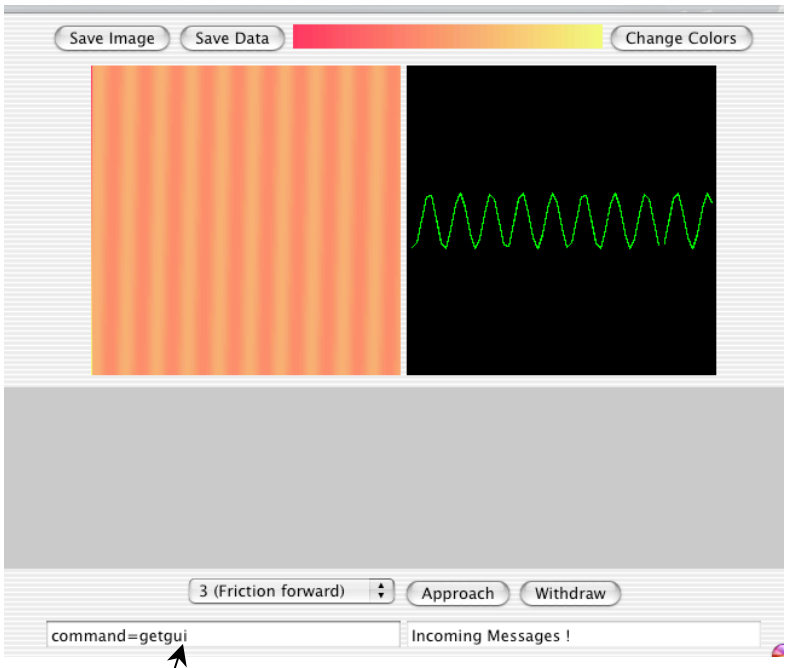
PART IV: Running

Unix:	Windows:
server.sh &	server.bat
client.sh 1313	client.bat 1313

Enter the following command sequence into the command window:

command=getgui

press enter



command window

the three controllers are pushed from the server to the client.