

Code Listings

Code Listings	1
Code for Scenario 1	1
weather_predictor.m	1
array_m2].m	2
IWeatherObject.java	3
WeatherObject.java	3
WeatherMonitor.java	4
WeatherData.java	4
Code for Scenario 2	5
input_machine.m	5
brightness_machine.m	5
monochrome_machine.m	6
output_machine.m	8
view_cimage.m	8
IBrightnessMachine.java	9
BrightnessMachine.java	9
IMonochromeMachine.java	10
MonochromeMachine.java	11
IOutputMachine.java	12
OutputMachine.java	13
ImageData.java	14
ImageDataMono.java	14
Code for Scenario 3	15
IManager.java	15
Manager.java	15
worker.m	18
starter.m	19
retriever.m	19
aborter.m	20
Slot.java	20
Results.java	21
Parameters.java	21

Code for Scenario 1

weather_predictor.m

```
%Matlab Distributed Computing Tutorial
%Scenario 1: Delivering the Weather Report
%Gabor Cselles, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselles

%weather_predictor.m:
% Matlab script for controlling the "weather predictions"
% and sending the results to the monitor

function [] = weather_predictor(weatherObjectHostname);

%updateNum gets increased each time we send results
updateNum = 1;

%set up initial data for simulation
city1name = 'Zurich';
city2name = 'New York';
city1weather = [20 50];
city2weather = [2 20];

weatherURL = strcat('rmi://',weatherObjectHostname,'/WO');

%connect to the weather object
try
    weatherObject = javaMethod('lookup','java.rmi.Naming',weatherURL);
catch
    disp(lasterr);
```

```

        disp('Some error ocured while contacting the remote WeatherObject. Aborting.');
```

return;

end

%loop calculation forever

```

while (1)
    unstr = int2str(updateNum);
    dispstr = strcat('Starting calculation number: ',unstr);
    disp(dispstr);

    size_c1w = size(city1weather);
    size_c2w = size(city2weather);

    %perform calculation
    [city1weather,city2weather] =
        weather_calc(city1weather(size_c1w(1,:),:),city2weather(size_c2w(1,:),:));

    disp (city1weather);
    disp (city2weather);

    %wrap results in Java WeatherData structure
    weatherData = javaObject('scen1.WeatherData');
    weatherData.city1name = city1name;
    weatherData.city2name = city2name;
    weatherData.city1weather = array_m2j(city1weather);
    weatherData.city2weather = array_m2j(city2weather);

    %do some more intensive "calculation"
    %(take this out for real applications!)
    pause(2);

    %send results to remoteObject
    try
        javaMethod('DeliverWeather',weatherObject,updateNum,weatherData);
    catch
        disp(lasterr);
        disp('Some error ocured while delivering the weather to the remote
            WeatherObject. Aborting.');
```

return;

end

updateNum = updateNum + 1;

end

%%%

%weather_calc.m:

% Function for predicting the weather.

```

function [city1results,city2results] = weather_calc(city1start,city2start);
%the start values are always the first prediction
city1results(1,1:2) = city1start(1,1:2);
city2results(1,1:2) = city2start(1,1:2);

%'predict' the weather
for i = 2:4
    %try to keep both temperatures between -10 and 40 degrees Celsius
    city1results(i,1) = city1results(i-1,1)+unifrnd(-(city1results(i-1,1) >
    -5),(city1results(i-1,1) < 35))*5;
    city2results(i,1) = city2results(i-1,1)+unifrnd(-(city2results(i-1,1) >
    -5),(city2results(i-1,1) < 35))*5;

    %probability of rain, keep between 0 and 100
    city1results(i,2) = city1results(i-1,2)+unifrnd(-(city1results(i-1,2) >
    5),(city1results(i-1,2) < 95))*5;
    city2results(i,2) = city2results(i-1,2)+unifrnd(-(city2results(i-1,2) >
    5),(city2results(i-1,2) < 95))*5;
end
end
```

array_m2j.m

```

function [J] = array_m2j(M);

%convert a 2-dimensional Matlab array to a java array
S = size(M);
J = javaArray('java.lang.Double', S(1), S(2));
```

```

for i = 1:S(1)
    for j = 1:S(2)
        J(i,j) = java.lang.Double(M(i,j));
    end
end
end

```

IWeatherObject.java

```

//Matlab Distributed Computing Tutorial
//Scenario 1: Delivering the Weather Report
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//IWeatherObject.java:
// Interface for WeatherObject

package scen1;
import java.lang.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public interface IWeatherObject extends Remote {
    public void TestCall() throws RemoteException;
    public void DeliverWeather(int updateNumber, WeatherData w) throws RemoteException;
}

```

WeatherObject.java

```

//Matlab Distributed Computing Tutorial
//Scenario 1: Delivering the Weather Report
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//WeatherObject.java:
// Remote object for weather predicition

package scen1;
import java.lang.*;
import java.text.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class WeatherObject extends UnicastRemoteObject implements IWeatherObject {
    public WeatherObject() throws RemoteException {
        super();
    }

    public void TestCall() throws RemoteException {
        //unnecessary utility function for testing the connection
        return;
    }

    public void DeliverWeather(int updateNumber, WeatherData w) throws RemoteException
    {
        //this function is used by the Matlab script to deliver the weather report
        System.out.println("Received weather update number " + updateNumber + ":");

        DecimalFormat df = new DecimalFormat("#0.00");

        System.out.println("City: " + w.city1name);

        for (int i = 0; i < w.city1weather.length; i++) {
            System.out.println(i + ": Temperature: " + df.format(w.city1weather[i][0]) +
                "\t Chance of rain: " + df.format(w.city1weather[i][1]) + "%");
        }

        System.out.println("City: " + w.city2name);

        for (int i = 0; i < w.city2weather.length; i++) {
            System.out.println(i + ": Temperature: " + df.format(w.city2weather[i][0]) +
                "\t Chance of rain: " + df.format(w.city2weather[i][1]) + "%");
        }
    }
}

```

WeatherMonitor.java

```
//Matlab Distributed Computing Tutorial
//Scenario 1: Delivering the Weather Report
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//WeatherMonitor.java:
// Main class of monitoring application.

package scen1;
import java.lang.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class WeatherMonitor {
    public static WeatherObject WO;

    public static void main(String args[]) {
        try {
            WO = new WeatherObject();
            Naming.rebind("WO", WO);
            System.out.println("WeatherObject ready.");

            System.out.println("=====");
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            return;
        }
    }
}
```

WeatherData.java

```
//Matlab Distributed Computing Tutorial
//Scenario 1: Delivering the Weather Report
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//WeatherData.java:
// Encapsulates all data that is to be delivered by the Matlab script

package scen1;
import java.io.*;

public class WeatherData implements Serializable {
    //these strings contain the name of the two cities we're predicting the weather for
    public String city1name;
    public String city2name;

    //these 2-dimensional arrays contain the weather prediction for the two cities
    //they have been converted from the weather prediction matrices of Matlab
    //the rows (1st dimension) correspond to the days we're predicting for
    //the cols (2nd dimension) are as follows:
    //      - 1st column: predicted temperature
    //      - 2nd column: chance of rain
    public Double[][] city1weather;
    public Double[][] city2weather;
}
```

Code for Scenario 2

input_machine.m

```
%Matlab Distributed Computing Tutorial
%Scenario 2: The Monochrome Image Converter
%Gabor Cselle, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselle

%input_machine.m:
% Matlab script for reading in all images as
% tensors and shooting them off to the brightness machine.

function [] = input_machine(BMHostname)

BMURL = strcat('rmi://',BMHostname,'/BrightnessMachine');

%connect to the Brightness machine object
try
    BM = javaMethod('lookup','java.rmi.Naming',BMURL);
catch
    disp(lasterr);
    disp('Some error occured while contacting the Brightness Machine. Aborting:');
    return;
end

%read in and shoot off photos 1 to 4
disp('Sending images 1-4');
[I,d1,d2,d3] = read_image('photo1.jpg');
send_image(I,d1,d2,d3,BM);

[I,d1,d2,d3] = read_image('photo2.jpg');
send_image(I,d1,d2,d3,BM);

[I,d1,d2,d3] = read_image('photo3.jpg');
send_image(I,d1,d2,d3,BM);

[I,d1,d2,d3] = read_image('photo4.jpg');
send_image(I,d1,d2,d3,BM);

%finished sending pictures, we can simply return
disp('Complete. ');
return;

%-----
%utility function to send images to brightness machines
function [] = send_image(I,d1,d2,d3,BM);

try
    BM.ReceiveImage(I,d1,d2,d3);
catch
    disp(lasterr);
    disp('Some error occured while sending an image to the remote Brightness Machine.
Aborting: ');
    return;
end

%-----
%utility function to read in images and
%return them as Java-compatible arrays
function [I,d1,d2,d3] = read_image(filename);

%efficient transfer of data from Matlab to Java:
%reshape the input tensor into a 1-dimensional array
%and pass it as such - no data conversion needed
Image = int16(imread(filename));
sizeImage = size(Image);
d1 = sizeImage(1);
d2 = sizeImage(2);
d3 = sizeImage(3);
I = reshape(Image,[],1);
```

brightness_machine.m

```
%Matlab Distributed Computing Tutorial
```

```

%Scenario 2: The Monochrome Image Converter
%Gabor Cselles, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselles

%brightness_machine.m:
% Matlab script for handling all the image brightness corrections.
% Continuously checks the Java object's input queue for incoming images.

function [] = brightness_machine(CMHostname);

%instantiate the brightness machine
BM = javaObject('scen2.BrightnessMachine',CMHostname);
%check if that went OK
if (BM.bConnected == 0)
    error('Brightness machine instantiation failed. Make sure the MonochromeMachine is
on and RMRegistry is running.');
```

```

    return;
end

while (1)
    if (javaMethod('ImagesWaiting',BM) > 0)
        disp('Getting image from queue');
        JImage = javaMethod('GetNextImage',BM);

        disp('Processing image ...');
        %bring the image into its original tensor form
        Image = reshape(JImage.data,JImage.dim1,JImage.dim2,JImage.dim3);

        %process image
        Image = adjust_brightness(Image);

        %reshape image for sending
        Image = reshape(Image,[],1);

        %send image out to the MonochromeMachine
        disp('Sending image to MonochromeMachine');
        BM.SendImage(Image,JImage.dim1,JImage.dim2,JImage.dim3);
    else
        pause(2); %give some time for the other processes to breathe
    end
end

end

%-----
%utility function to perform the brightness adjustments
function [I] = adjust_brightness(I);

I = double(I);
I(:,:,:) = min(I(:,:,:)+30,255); %increase brightness everywhere
I = int16(I);

```

monochrome_machine.m

```

%Matlab Distributed Computing Tutorial
%Scenario 2: The Monochrome Image Converter
%Gabor Cselles, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselles

%monochrome_machine.m:
% Matlab script the conversion to a monochrome image.
% Same concept as the brightness machine's script.

function [] = monochrome_machine(OMHostname);

%instantiate the monochromization machine
CM = javaObject('scen2.MonochromeMachine',OMHostname);
%check if that went OK
if (CM.bConnected == 0)
    error('Monochrome machine instantiation failed. Make sure the OutputMachine is on
and RMRegistry is running.');
```

```

    return;
end

while (1)
    if (javaMethod('ImagesWaiting',CM) > 0)
        disp('Getting image from queue');
        JImage = javaMethod('GetNextImage',CM);

```

```

disp('Processing image ...');
%bring the image into its original tensor form
Image = reshape(JImage.data,JImage.dim1,JImage.dim2,JImage.dim3);

%process image
[Image,Vmax,mu] = monochromize_image(Image);

%reshape image for sending
Image = reshape(Image,[],1);
Vmax = reshape(Vmax,[],1);
mu = reshape(mu,[],1);

%send image out to the OutputMachine
disp('Sending image to OutputMachine');
CM.SendImage(Image,JImage.dim1,JImage.dim2,JImage.dim3,Vmax,mu);
else
    pause(2); %give some time for the other processes to breathe
end
end

%-----
%utility function to perform the "monochromisation"
%this code was inspired by an exercise given in the course
%"Numerical and Symbolic Computation" by Prof. Petros Koumoutsakos
% in WS2002/2003 at ETH Zurich
%(you can safely ignore this code as it does not have any significance
% for the distributed Matlab material)
function [C,Vmax,mu] = monochromize_image(I);

nEig = 1; %take only the first color space eigenvector

S = [];

%1. convert input matrix to doubles
I = double(I);
%2. calculate matrix channel covariance matrix S
for i = 1:3
    %calculate median
    mu(i) = mean(mean(I(:,:,i)));
    %center data
    I(:,:,i) = I(:,:,i) - mu(i);
end
for i = 1:3
    for j = 1:3
        S(i,j) = mean(mean((I(:,:,i)) .* (I(:,:,j))));
    end
end
%3. determine eigenvalues and eigenvectors of covariance matrix
[V,D] = eig(S);
%take out the nEig largest eigenvectors, store into Vmax
Vmax = []; maxE = 0; maxEi = 0;
dDiag = diag(D);
for e = 1:nEig
    maxE = 0; %value of largest eigenvector
    maxEi = 0; %index to largest eigenvector
    for i = 1:3
        if abs(dDiag(i)) > abs(maxE)
            maxE = dDiag(i);
            maxEi = i;
        end;
    end
    %make sure the eigenvector never appears again
    dDiag(maxEi) = 0;
    %put the new eigenvector into the Vmax
    Vmax(:,e) = V(:,maxEi);
end
Isize = size(I);
%4. calculate the compressed data
for i = 1:Isize(1)
    for j = 1:Isize(2)
        %assemble y vector
        y(1,1) = I(i,j,1); y(2,1) = I(i,j,2); y(3,1) = I(i,j,3);
        C(i,j,:) = Vmax'*y;
    end
end
end
return;

```

output_machine.m

```
%Matlab Distributed Computing Tutorial
%Scenario 2: The Monochrome Image Converter
%Gabor Cselle, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselle

%output_machine.m:
% Matlab script for the output machine.
% Keeps checking the Java object's input queue and then stores
% the images to disk as Matlab tensors.

function [] = monochromization_machine();

n_images = 0; %number of images received

%instantiate the output machine
OM = javaObject('scen2.OutputMachine');
%check if that went OK
if (OM.bConnected == 0)
    error('Output machine instantiation failed. Make sure RMIregistry is running.');
```

```
    return;
end

while (1)
    if (javaMethod('ImagesWaiting',OM) > 0)
        disp('Getting image from queue');
        JImage = javaMethod('GetNextImage',OM);

        disp('Processing image ...');
        %bring the monochrome image into its original tensor form
        CImage = reshape(JImage.data,JImage.dim1,JImage.dim2,[]);

        %bring the helper matrices into their correct form
        %(Vmax and mu always have the same form so we don't need to
        %remember their dimensions)
        Vmax = reshape(JImage.Vmax,3,[]);
        mu = reshape(JImage.mu,1,[]);

        %send image out to the OutputMachine
        disp('Storing the monochrome image into:');
        %assemble the file name
        n_images = n_images+1;
        filename = strcat('monoimg',int2str(n_images),'.mat');
        save(filename,'CImage','Vmax','mu');
        disp(filename);
    else
        pause(2); %give some time for the other processes to breathe
    end
end
```

view_cimage.m

```
%Matlab Distributed Computing Tutorial
%Scenario 2: The Monochrome Image Converter
%Gabor Cselle, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselle

%view_cimage.m:
% Matlab script for viewing the monochrome images.
% Just for fun.

function [0] = view_cimage(filename);

load(filename);

nEig = 1; %the number of eigenvectors shall be 1 here
        %(just like at the monochromization stage)

Csize = size(CImage);

%1. decompress data
for i = 1:Csize(1)
    for j = 1:Csize(2)
        for k = 1:nEig
            c(k,1) = CImage(i,j,k);
```

```

        end
        O(i,j,1:3) = Vmax * c;
    end
end
2. %re-align data with the median
for i = 1:3
    O(:, :, i) = O(:, :, i) + mu(i);
end
%3. convert the data back to unsigned integers
O = uint8(O);

```

IBrightnessMachine.java

```

//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//IBrightnessMachine.java:
// Interface for BrightnessMachine class.

package scen2;
import java.lang.*;
import java.rmi.*;
import java.rmi.server.*;

public interface IBrightnessMachine extends Remote {
    public void TestCall() throws RemoteException;
    public void ReceiveImage(short[] data, int dim1, int dim2, int dim3)
        throws RemoteException;
}

```

BrightnessMachine.java

```

//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//BrightnessMachine.java:
// Java object handling incoming queries from the input machine.
// Incoming data is put into a queue, which will be processed by the
// Matlab script. Also handles outgoing communication to the monochromization
// machine.

package scen2;
import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class BrightnessMachine extends UnicastRemoteObject implements
IBrightnessMachine {
    private String CMHostname;
    private IMonochromeMachine CM; //the remote MonochromeMachine object
    public boolean bConnected = false;
    private LinkedList InputQueue; //LinkedList that will hold our input queue

    public BrightnessMachine(String CMHostname) throws RemoteException {
        //this constructor will:
        // - make an instance of the BrightnessMachine as a remote object,
        // - register itself with the RMIregistry naming lookup process, and
        // - connect to the monochromization machine with hostname CMHostname
        super();
        try {
            Naming.rebind("BrightnessMachine", this);
            System.out.println("BrightnessMachine ready.");
        } catch (Exception e) {
            System.out.println("Exception: " + e);
            return;
        }
    }

    //connect to MonochromeMachine
    this.CMHostname = CMHostname;
    try {
        System.out.println("Contacting MonochromeMachine at " + "rmi://" +

```

```

        CMHostname + "/" + "MonochromeMachine");
        CM = (IMonochromeMachine)Naming.lookup("rmi://" + CMHostname + "/"
        + "MonochromeMachine");
    } catch (NotBoundException e) {
        bConnected = false;
        System.out.println("ERROR: Looking up MonochromeMachine failed!");
        System.out.println(e);
    } catch (RemoteException e) {
        bConnected = false;
        System.out.println("ERROR: Contacting MonochromeMachine failed!");
        System.out.println(e);
    } catch (Exception e) {
        bConnected = false;
        System.out.println("ERROR when starting MonochromeMachine!");
        System.out.println(e);
    }

    //create input queue
    InputQueue = new LinkedList();

    bConnected = true;

    return;
}

public void TestCall() throws RemoteException {
    //unnecessary utility function for testing the connection
    return;
}

public int ImagesWaiting() {
    //utility function for Matlab to find out if there are any images waiting
    //to be processed (does not throw a RemoteException; you should not be able to
    //call this remotely)
    return InputQueue.size();
}

public ImageData GetNextImage() {
    //function to pop the next waiting image from the FIFO list
    //(does not throw a RemoteException; you should not be able to call this
    //remotely)
    return (ImageData)InputQueue.removeFirst();
}

public void SendImage(short[] data, int dim1, int dim2, int dim3) {
    //sends image to remote Monochrome Machine
    //(does not throw a RemoteException; you should not be able to call this
    //remotely)
    try {
        CM.ReceiveImage(data, dim1, dim2, dim3);
    } catch (RemoteException e) {
        System.out.println("ERROR: at SendImage: Communication with
        MonochromeMachine failed!");
        System.out.println(e);
    }
    return;
}

public void ReceiveImage(short[] data, int dim1, int dim2, int dim3)
throws RemoteException {
    //this function is used by the input machine to deliver new images
    System.out.println("Received image. Adding to queue.");
    ImageData image = new ImageData();
    image.data = data; image.dim1 = dim1; image.dim2 = dim2; image.dim3 = dim3;
    //add the image at the end of the (FIFO) queue
    InputQueue.addLast((Object)image);
    return;
}
}
}

```

IMonochromeMachine.java

```

//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch

```

```
//(C) 2004 Gabor Cselle

//IMonochromeMachine.java:
// Interface for thhe MonochromeMachine class.

package scen2;
import java.lang.*;
import java.rmi.*;
import java.rmi.server.*;

public interface IMonochromeMachine extends Remote {
    public void TestCall() throws RemoteException;
    public void ReceiveImage(short[] data, int dim1, int dim2, int dim3)
        throws RemoteException;
}

```

MonochromeMachine.java

```
//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//MonochromeMachine.java:
// Java object for the monochrome machine.
// Same concept as the brightness machine's java object.

package scen2;
import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class MonochromeMachine extends UnicastRemoteObject implements
IMonochromeMachine {
    private String OMHostname;
    private IOutputMachine OM; //the remote MonochromeMachine object
    public boolean bConnected = false;
    private LinkedList InputQueue; //LinkedList that will hold our input queue

    public MonochromeMachine(String OMHostname) throws RemoteException {
        //this constructor will:
        // - create an instance of the MonochromeMachine as a remote object,
        // - register itself with the RMIregistry naming lookup process, and
        // - connect to the monochromization machine with hostname CMHostname
        super();
        try {
            Naming.rebind("MonochromeMachine", this);
            System.out.println("MonochromeMachine ready.");
        } catch (Exception e) {
            System.out.println("Exception: " + e);
            return;
        }
    }

    this.OMHostname = OMHostname;

    try {
        System.out.println("Contacting OutputMachine at " + "rmi://" + OMHostname
            + "/" + "OutputMachine");
        OM = (IOutputMachine)Naming.lookup("rmi://" + OMHostname + "/" +
            "OutputMachine");
    } catch (NotBoundException e) {
        bConnected = false;
        System.out.println("ERROR: Looking up OutputMachine failed!");
        System.out.println(e);
    } catch (RemoteException e) {
        bConnected = false;
        System.out.println("ERROR: Contacting OutputMachine failed!");
        System.out.println(e);
    } catch (Exception e) {
        bConnected = false;
        System.out.println("ERROR when starting OutputMachine!");
        System.out.println(e);
    }
}

```

```

        //create input queue
        InputQueue = new LinkedList();

        bConnected = true;

        return;
    }

    public void TestCall() throws RemoteException {
        //unnecessary utility function for testing the connection
        return;
    }

    public int ImagesWaiting() {
        //utility function for Matlab to find out if there are any images waiting
        //to be processed (does not throw a RemoteException; you should not be able
        //to call this remotely)
        return InputQueue.size();
    }

    public ImageData GetNextImage() {
        //function to pop the next waiting image from the FIFO list
        //(does not throw a RemoteException; you should not be able to call this
        //remotely)
        return (ImageData)InputQueue.removeFirst();
    }

    public void SendImage(double[] data, int dim1, int dim2, int dim3, double[] Vmax,
        double[] mu) {
        //sends image to remote Output Machine
        //(does not throw a RemoteException; you should not be able to call this
        //remotely)
        try {
            OM.ReceiveImage(data, dim1, dim2, dim3, Vmax, mu);
        } catch (RemoteException e) {
            System.out.println("ERROR: at SendImage: Communication with MonochromeMachine
                failed!");
            System.out.println(e);
        }
        return;
    }

    public void ReceiveImage(short[] data, int dim1, int dim2, int dim3)
        throws RemoteException {
        //this function is used by the input machine to deliver new images
        System.out.println("Received image. Adding to queue.");
        ImageData image = new ImageData();
        image.data = data; image.dim1 = dim1; image.dim2 = dim2; image.dim3 = dim3;
        //add the image at the end of the (FIFO) queue
        InputQueue.addLast((Object)image);
        return;
    }
}

```

IOutputMachine.java

```

//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

```

```

//IOutputMachine.java:
// Interface for OutputMachine class.

```

```

package scen2;
import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public interface IOutputMachine extends Remote {
    public void TestCall() throws RemoteException;
    public void ReceiveImage(double[] data, int dim1, int dim2, int dim3,
        double[] Vmax, double[] mu) throws RemoteException;
}

```

OutputMachine.java

```
//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//OutputMachine.java:
// Java object for the output machine.
// Handles the input queue, which is filled by the monochromization machine.

package scen2;
import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class OutputMachine extends UnicastRemoteObject implements IOutputMachine {
    public boolean bConnected = false;
    private LinkedList InputQueue; //LinkedList that will hold our input queue

    public OutputMachine() throws RemoteException {
        //this constructor will:
        // - create an instance of the OutputMachine as a remote object,
        // - register itself with the RMIregistry naming lookup process
        super();
        try {
            Naming.rebind("OutputMachine", this);
            System.out.println("OutputMachine ready.");
        } catch (Exception e) {
            System.out.println("Exception: " + e);
            return;
        }

        //create input queue
        InputQueue = new LinkedList();

        bConnected = true;

        return;
    }

    public void TestCall() throws RemoteException {
        //unnecessary utility function for testing the connection
        return;
    }

    public int ImagesWaiting() {
        //utility function for Matlab to find out if there are any images waiting to be
        //processed(does not throw a RemoteException; you should not be able to call
        //this remotely)
        return InputQueue.size();
    }

    public ImageDataMono GetNextImage() {
        //function to pop the next waiting image from the FIFO list
        //(does not throw a RemoteException; you should not be able to call this
        //remotely)
        return (ImageDataMono)InputQueue.removeFirst();
    }

    public void ReceiveImage(double[] data, int dim1, int dim2, int dim3, double[]
        Vmax, double[] mu) throws RemoteException {
        //this function is used by the input machine to deliver new images
        System.out.println("Received image. Adding to queue.");
        ImageDataMono image = new ImageDataMono();
        image.data = data; image.dim1 = dim1; image.dim2 = dim2; image.dim3 = dim3;
        image.Vmax = Vmax; image.mu = mu;
        //add the image at the end of the (FIFO) queue
        InputQueue.addLast((Object)image);
        return;
    }
}
}
```

ImageData.java

```
//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//ImageData.java:
// Simple class for holding image data in the input queues.

package scen2;

public class ImageData {
    public short[] data;
    public int dim1;
    public int dim2;
    public int dim3;
}
```

ImageDataMono.java

```
//Matlab Distributed Computing Tutorial
//Scenario 2: The Monochrome Image Converter
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//ImageDataMono.java:
// Simple class for holding monochrome image data in the
// output machine's input queue.

package scen2;

public class ImageDataMono {
    public double[] data;
    public int dim1;
    public int dim2;
    public int dim3;
    public double[] Vmax;
    public double[] mu;
}
```

Code for Scenario 3

IManager.java

```
//Matlab Distributed Computing Tutorial
//Scenario 3: Simulating Normal Distributions
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//IManager.java:
// Interface for Manager class.

package scen3;
import java.lang.*;
import java.util.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public interface IManager extends Remote {
    public void TestCall() throws RemoteException;
    public Slot RequestSlot() throws RemoteException;
    public void DeliverSlot(Slot s) throws RemoteException;
    public void StartCalculation(Parameters p) throws RemoteException;
    public Results RetrieveResults() throws RemoteException;
    public void AbortCalculation() throws RemoteException;
}
```

Manager.java

```
//Matlab Distributed Computing Tutorial
//Scenario 3: Simulating Normal Distributions
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//Manager.java:
// Remote object for the manager of the distributed calculation

package scen3;
import java.lang.*;
import java.io.*;
import java.rmi.*;
import java.rmi.server.*;

public class Manager extends UnicastRemoteObject implements IManager {
    //keep track of the state of calculations
    private boolean bCalculating = false;
    private boolean bHaveResults = false;

    //data for current calculation
    private int[] slotsAssignment = new int[0]; //remembers whether the slots state:
    //-1: unassigned; 0: assigned, but not completed; 1: assigned and completed

    private Slot[] slots = new Slot[0]; //stores the slots of the current calculation
    private Parameters parameters = new Parameters();
    //parameters for the current calculation

    //////////////////////////////////////
    //Manager initializer, constructor, test call

    //allow the Manager as a stand-alone
    public static void main(String args[]) {
        Manager manager;
        try {
            manager = new Manager();
            Naming.rebind("Manager", manager);
            System.out.println("Manager ready.");
            System.out.println("=====");
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            return;
        }
    }

    public Manager() throws RemoteException {
        super();
    }
}
```

```

    return;
}

public void TestCall() throws RemoteException {
    //unnecessary utility function for testing the connection
    return;
}

////////////////////////////////////
//Interface functions to be called by clients

public void StartCalculation(Parameters p) throws RemoteException {
    //start the calculation with a certain set of parameters

    //make sure we're not in the middle of a calculation
    if (!bCalculating) {
        //store the parameters locally
        this.parameters = p;

        //initialize the slots
        SlotsAllocateUnassigned(this.parameters.totalSlots);

        //set the state
        bCalculating = true;
        bHaveResults = false;

        //print something for the user
        System.out.println("===== " );
        System.out.println("Calculation STARTED" );
        System.out.println("-----" );
    } else {
        System.out.println("ERROR:
            Tried to start a calculation while we're still running another!");
    }

    return;
}

public Slot RequestSlot() throws RemoteException {
    //returns a slot object with data to be processed or an empty slot object if
there is nothing to do
    Slot slot = new Slot(); //the response object

    //try to find an unassigned slot and assign it
    //(this needs to be done in a single operation to avoid anyone else
    int slotNum = SlotAssignUnassigned();

    if (slotNum != -1) {
        //We found an unassigned slot

        //REQUIRED
        slot.slotNum = slotNum;
        slot.totalSlots = parameters.totalSlots;
        slot.bWait = false;

        //APPLICATION-SPECIFIC
        slot.numSims = parameters.numSims;
        slot.mean = parameters.means[slotNum];
        //leave the means array alone, it will be filled by the Worker

        System.out.println("Assigned slot: " + slotNum);
    } else {
        //if we can't find an unassigned slot, return a Slot object that says "wait!"
        slot.bWait = true;
    }

    return slot;
}

public void DeliverSlot(Slot s) throws RemoteException {
    //delivery of a processed slot by the Worker

    //make sure we are currently accepting slots
    if (!bCalculating) {

```

```

        System.out.println("Warning: worker tried to deliver slots while no
calculation was running!" );
        return;
    }

    //take the slot number out of the slot and store it
    int slotNum = s.slotNum;
    slots[slotNum] = s;

    //mark the slot as delivered
    boolean bFinished = SlotMarkDelivered(slotNum);

    //if the calculation is finished, put us in the respective state
    if (bFinished) {
        this.bCalculating = false;
        this.bHaveResults = true;
        System.out.println("-----");
        System.out.println("Calculation FINISHED!" );
        System.out.println("=====");
    }

    return;
}

public Results RetrieveResults() throws RemoteException {
    //returns the results of the current calculation or an empty Results object if
the calculation is not yet finished.
    Results results = new Results();

    //check if we even have results we can deliver and are not currently calculating
    if (bHaveResults && (!bCalculating)) {
        System.out.println("Delivering results ..." );
        results.bWait = false;
        results.slots = slots;
        results.totalSlots = parameters.totalSlots;
    } else {
        //tell the retriever to wait
        results.bWait = true;
    }

    return results;
}

public void AbortCalculation() throws RemoteException {
    //aborts the current calculation, no matter what the state is
    this.parameters = null;
    this.bCalculating = false;
    this.bHaveResults = false;

    SlotsAllocateUnassigned(0);

    System.out.println("-----");
    System.out.println("Calculation ABORTED!" );
    System.out.println("=====");

    return;
}

////////////////////////////////////
//Private utility functions,
//mostly handling the SlotAssignment array.
//All of these functions need to be synchronized, so they
//can't interfere which other when writing to the array

public synchronized void SlotsAllocateUnassigned(int totalSlots) {
    slotsAssignment = new int[totalSlots];
    slots = new Slot[totalSlots];
    for (int i = 0; i < totalSlots; i++) {
        slotsAssignment[i] = -1; //set all slots to unassigned
    }
}

public synchronized int SlotAssignUnassigned() {
    //assign stationNo to the first unassigned slot
    int i = 0;

    while (i < parameters.totalSlots) {

```

```

        if (slotsAssignment[i] == -1) {
            slotsAssignment[i] = 0;

            return i;
        }
        i++;
    }

    return -1;
}

public synchronized boolean SlotMarkDelivered(int slotNum) {
    slotsAssignment[slotNum] = 1;

    //at this point, the calculation might be finished
    for (int i = 0; i < parameters.totalSlots; i++) {
        if (slotsAssignment[i] <= 0) {
            return false;
        }
    }

    return true;
}
}

```

worker.m

```

%Matlab Distributed Computing Tutorial
%Scenario 3: Simulating Normal Distributions
%Gabor Cselle, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselle

```

```

%worker.m:

```

```

% Matlab script for the workers.
% When it has nothing to do, it is stuck in a loop and has to ask the
% Manager at a randomized interval for new slots to process.

```

```

function [] = worker(manager_hostname);

```

```

rand('state',sum(100*clock));
waittime = 10 + unifrnd(-2,2);
disp('My randomized waiting time: ');
disp(waittime);

```

```

managerObject = [];

```

```

managerURL = strcat('rmi://',manager_hostname,'/Manager');

```

```

%connect to the managerObject

```

```

try
    managerObject = javaMethod('lookup','java.rmi.Naming',managerURL);
catch
    disp(lasterr);
    disp('Some error occured while contacting the Manager. Aborting:');
    return;
end

```

```

disp('Starting calculations');

```

```

%this is a starter, loop endlessly

```

```

while (1)
    %request a slot
    slot = javaMethod('RequestSlot',managerObject);

    if (slot.bWait)
        %we didn't get a slot, wait for some time
        pause(waittime);
    else
        disp(strcat('Processing slot number: ',int2str(slot.slotNum)));
        %APPLICATION-SPECIFIC
        %we did get a slot, process it
        slot.sims = process_slot(slot.slotNum,slot.totalSlots,slot.numSims,slot.mean);
    end
end

```

```

    pause(5);
    %return the slot
    javaMethod('DeliverSlot',managerObject,slot);
end
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5
% local processing function
% create the normally distributed functions we wanted
function [sims] = process_slot(slot_num,total_slots,num_sims,mean);

sims = normrnd(mean,1,1,num_sims);

return;

```

starter.m

```

%Matlab Distributed Computing Tutorial
%Scenario 3: Simulating Normal Distributions
%Gabor Cselle, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselle

%starter.m:
% A Matlab script for starting the calculations.
% The normal distribution means have to be specified in the parameters.
function [] = starter(manager_hostname,means,simsPerSlot);

managerObject = [];

managerURL = strcat('rmi://',manager_hostname,'/Manager');

%connect to the managerObject
try
    managerObject = javaMethod('lookup','java.rmi.Naming',managerURL);
catch
    disp(lasterr);
    disp('Some error occured while contacting the Manager. Aborting:');
    return;
end

%reshape the means so they're in the correct format
means = reshape(means,[],1);
totalSlots = length(means);

%create a parameters object and fill in the values
p = javaObject('scen3.Parameters');
p.totalSlots = totalSlots;
p.means = means;
p.numSims = int32(simsPerSlot);

%send off the parameters and start the calculation
try
    javaMethod('StartCalculation',managerObject,p);
catch
    disp(lasterr);
    disp('Some error occured while delivering parameters to the Manager. Aborting.');
```

retriever.m

```

%Matlab Distributed Computing Tutorial
%Scenario 3: Simulating Normal Distributions
%Gabor Cselle, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselle

%retriever.m:
% Matlab script to retrieve the results of the last calculation.
% Returns a Boolean variable which is true if it could fetch
% the results (or false if it could not because the calculation has not yet
% finished).

function [flag,data] = retriever(manager_hostname);

managerObject = [];

managerURL = strcat('rmi://',manager_hostname,'/Manager');

%connect to the managerObject
try
    managerObject = javaMethod('lookup','java.rmi.Naming',managerURL);
catch

```

```

        disp(lasterr);
        disp('Some error occured while contacting the Manager. Aborting:');
        return;
    end

%now, retrieve the results
disp('Retrieving results ...');
try
    results = javaMethod('RetrieveResults',managerObject);
catch
    disp(lasterr);
    disp('Some error occured while communicating with the Manager. Aborting. ');
    return;
end
disp('Results recieved. ');

if (results.bWait)
    disp('The last calculation has not yet finished!');
    flag = 0;
    return;
end

flag = 1;
data = [];

%put all of the results together
totalSlots = results.totalSlots;
for i = 1:totalSlots;
    currSlot = results.slots(i);
    currData = currSlot.sims;
    data = [data; currData];
end

```

aborter.m

```

%Matlab Distributed Computing Tutorial
%Scenario 3: Simulating Normal Distributions
%Gabor Cselle, gabor(at)student.ethz.ch
%(C) 2004 Gabor Cselle

%aborter.m:
% Matlab script to abort the calculation that's currently running

function [flag,data] = aborter(manager_hostname);

%this will create a Starter object and connect to the ManagedObject
managerObject = [];

managerURL = strcat('rmi://',manager_hostname,'/Manager');

%connect to the managerObject
try
    managerObject = javaMethod('lookup','java.rmi.Naming',managerURL);
catch
    disp(lasterr);
    disp('Some error occured while contacting the Manager. Aborting:');
    return;
end

%now, retrieve the results
disp('Aborting current calculation ...');
try
    javaMethod('AbortCalculation',managerObject);
catch
    disp(lasterr);
    disp('Some error occured while communicating with the Manager. Aborting. ');
    return;
end

```

Slot.java

```

//Matlab Distributed Computing Tutorial
//Scenario 3: Simulating Normal Distributions
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

//Slot.java:

```

```

// Class containing the data for a slot

package scen3;
import java.lang.*;
import java.io.*;

public class Slot implements Serializable {
    //REQUIRED
    public int slotNum;    //number of this slot
    public int totalSlots; //out of a total number of slots

    public boolean bWait; //this is set to true if there is no currently unassigned
slot
                        //and the Worker needs to wait a bit longer

    //APPLICATION-SPECIFIC
    public int numSims;    //number of simulation that need to be carried out in this
slot
    public double mean;    //the mean of the normal distrubition

    public double[] sims; //results we got back from the simulation
                        //(when the Manager sends this out the Worker, this is empty)
}

```

Results.java

```

//Matlab Distributed Computing Tutorial
//Scenario 3: Simulating Normal Distributions
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

```

```

//Results.java:
// Class containing the results of a calculation

```

```

package scen3;
import java.lang.*;
import java.io.*;

public class Results implements Serializable {
    //REQUIRED
    public boolean bWait; //boolean variable signalling to wait
                        //when the calculation has not yet completed
    public int totalSlots; //total number of slots
    public Slot[] slots; //the resulting slots
}

```

Parameters.java

```

//Matlab Distributed Computing Tutorial
//Scenario 3: Simulating Normal Distributions
//Gabor Cselle, gabor(at)student.ethz.ch
//(C) 2004 Gabor Cselle

```

```

//Parameters.java:
// Class containing the initial parameters of a calculation

```

```

package scen3;
import java.lang.*;
import java.io.*;

public class Parameters implements Serializable {
    //REQUIRED
    public int totalSlots; //total number of slots that need to be carried
                        //out for this calculation

    //APPLICATION-SPECIFIC
    public int numSims;
        //number of simulations that need to be carried out in this slot
    public double[] means;
        //means ( $\mu$ s) of the normal distributions
}

```