```
mkdir bin
mkdir bin\tracker
mkdir bin\server
mkdir bin\client
cd src\tracker
call build.bat
if %errorlevel% neq 0 exit /b %errorlevel%
copy bin\tracker.jar ..\..\bin\tracker\
cd ..\
cd server
call build.bat
if %errorlevel% neq 0 exit /b %errorlevel%
copy bin\server.jar ..\..\bin\server\
cd ..\
cd client
call build.bat
if %errorlevel% neq 0 exit /b %errorlevel%
copy bin\client.jar ..\..\bin\client\
cd ..\..\

REM Copy Lib files
copy /Y lib\*.jar bin\server\
copy /Y tools\ffmpeg.exe bin\server\
set -e
mkdir -p bin
mkdir -p bin/tracker
mkdir -p bin/server
mkdir -p bin/client
cd src/tracker
./build.sh
cp bin/tracker.jar ../../bin/tracker/
cd ../
cd server
./build.sh
cp bin/server.jar ../../bin/server/
cd ../
cd client
./build.sh
cp bin/client.jar ../../bin/client/
cd ../../

# Copy Lib files
cp lib/*.jar bin/server/
cp tools/ffmpeg bin/server/
```

```
$ErrorActionPreference = "Stop"
Import-Module BitsTransfer
if(!(Test-Path -Path "bin\server\models\ssd_inception_v2_coco_2017_11_17\")){
  new-item -Name bin\server\models -Force -ItemType directory
  Start-BitsTransfer -Source "http://download.tensorflow.org/models/object_detec
tion/ssd_inception_v2_coco_2017_11_17.tar.gz" -Destination "ssd_inception_v2_coc
o_2017_11_17.tar.gz"
  .\tools\7za.exe e "ssd_inception_v2_coco_2017_11_17.tar.gz"
  .\tools\7za.exe x "ssd_inception_v2_coco_2017_11_17.tar" -obin\server\models\
-r
  Remove-Item -Path ssd_inception_v2_coco_2017_11_17.tar
  Remove-Item -Path ssd_inception_v2_coco_2017_11_17.tar.gz
}
if(!(Test-Path "bin\server\labels\mscoco_label_map.pbtxt" -PathType Leaf)){
  new-item -Name bin\server\labels -Force -ItemType directory
  Start-BitsTransfer -Source "https://raw.githubusercontent.com/tensorflow/model
s/865c14c1209cb9ae188b2a1b5f0883c72e050d4c/research/object_detection/data/mscoco
_label_map.pbtxt" -Destination  "bin\server\labels\mscoco_label_map.pbtxt"
}
if(!(Test-Path "bin\server\labels\oid_bbox_trainable_label_map.pbtxt" -PathType
Leaf)){
  new-item -Name bin\server\labels -Force -ItemType directory
  Start-BitsTransfer -Source "https://raw.githubusercontent.com/tensorflow/model
s/865c14c1209cb9ae188b2a1b5f0883c72e050d4c/research/object_detection/data/oid_bb
ox_trainable_label_map.pbtxt" -Destination  "bin\server\labels\oid_bbox_trainabl
e_label_map.pbtxt"
}
```

```
#!/bin/bash
set -e
if [ ! -d "bin/server/models/ssd_inception_v2_coco_2017_11_17/" ]; then
  mkdir -p bin/server/models/
  curl -L http://download.tensorflow.org/models/object_detection/ssd_inception_v
2_coco_2017_11_17.tar.gz | tar -xz -C bin/server/models/
fi
if [ ! -f "bin/server/labels/mscoco_label_map.pbtxt" ]; then
  mkdir -p bin/server/labels/
  curl -L -o bin/server/labels/mscoco_label_map.pbtxt "https://raw.githubusercontent.com/te
nsorflow/models/865c14c1209cb9ae188b2a1b5f0883c72e050d4c/research/object_detection/data/mscoco_label_map.p
btxt"
  curl -L -o bin/server/labels/oid_bbox_trainable_label_map.pbtxt "https://raw.githubu
sercontent.com/tensorflow/models/865c14c1209cb9ae188b2a1b5f0883c72e050d4c/research/object_detection/data/oid_
bbox_trainable_label_map.pbtxt"
fi
```

```
<style>
  code{
    color: #f92672 !important;
    border: 1px solid hsla(0, 0%, 89%, 1);
    background-color: hsla(0, 0%, 98%, 1);
    font-size: .875rem;
    border-radius: 2px;
    display: inline-block;
    padding: 3px 7px;
  }
</style>

# Distributed Processing Program

## Prerequisite

1. java >1.8
2. Make sure that java and javac is in path
3. Tensor flow (if want to run with simulation off) (Installation instruction is
 below)

## Build

1. Run `build.bat`

## Install Tensor flow models

If you pull from github, no download is needed. Skip all step to [Run](#run) sec
tion.
One of the service provided is to run Tensor Flow Object Detection. This will re
quire the tensor flow pre-trained models and labels. To download them run `downl
oad_models.ps1` or `download_models.sh`. To save time, you can also turn on simu
lation so that the it does not actually run the tensor flow command but return a
 hard coded value. See configuration below to find out how to turn it on.

## Configuration

If a configuration file is not found. First startup will create default config f
ile.

### Tracker

File: tracker.config.properties
| Config            | Definition        | Default |
| ----------------- | ----------------- | ------- |
| rmi_registry_port | Tracker RMI Port  | 1099    |

### Server

File: server.config.properties
| Config                    | Definition
                                                                         | Def
ault                                                      |
| ------------------------- | -----------------------------------------------
---------------------------------------------------------------------- | ---
---------------------------------------------------------- |
| image_analytics_simulate  | Turn simulate on or off see [Install Tensor flow m
odels](#install-Tensor-flow-models)                                      | 0
                                                          |
| services                  | Services that this server will provide. Split by c
omma (,)                                                                 | Vid
eoAnalytics,VideoSplit,ImageAnalytics,ImageAnalyticsGraph |
```

```
| image_analytics_label     | The label file for tensor flow. Not needed if simu
lation is 1. Or if the server is not providing ImageAnalyticsService.     | lab
els/mscoco_label_map.pbtxt                                |
| image_analytics_model_dir | The model directory for tensor flow. Not needed if
 simulation is 1. Or if the server is not providing ImageAnalyticsService. | mod
els/ssd_inception_v2_coco_2017_11_17/saved_model          |
| tracker                   | The tracker server
                                                                         | loc
alhost\:1099                                              |
| rmi_registry_port         | The server rmi registry port
                                                                         | 100
0                                                         |
| rmi_registry_host         | The server rmi registry host/IP
                                                                         | loc
alhost                                                    |
| ffmpeg_command            | The server ffmpeg command to run. Not needed if no
t providing VideoSplitService.                                           | ./f
fmpeg                                                     |

### Client

File: server.config.properties
| Config            | Definition                                         | Def
ault          |
| ----------------- | -------------------------------------------------- | ---
----------- |
| rmi_registry_port | Callback RMI Registry Port                         | 108
8             |
| rmi_registry_host | Callback RMI Registry Host/IP                      | loc
alhost        |
| trackers          | Tracker servers to find servers. Comma seperated (,) | loc
alhost:1099 |

## Run

### Run Tracker
Go to `bin\tracker` and run `java -jar tracker.jar` in command line / terminal

### Run Server
Go to `bin\server` and run `java -cp server.jar;xchart-3.5.2.jar Server` for win
dows in command line. Run `java -cp server.jar:xchart-3.5.2.jar Server` for linu
x in terminal

### Run Client
Go to `bin\client` and run `java -jar client.jar` in command line / terminal

# Acknowledgements & License

## TensorFlow demo Program

The ImageAnalytics potion of the program uses an external jar application to do
the analytics. Mainly the TensorFlow demo program that can be found at <https://
github.com/tensorflow/models/tree/master/samples/languages/java/object_detection
>

Since the jar demo application does not provide an easy way to extract info out,
 we modified the code to output a CSV file. It's then package as a standalone JAR file.
A copy of the license can be found at lib/detect-object-LICENSE

## Xchart

The ImageAnalyticsGraph part of the program uses an external library called xchart <https://knowm.org/open-source/x
```

chart/>. No modification is made on the source. A copy of the license can be found at lib/xchart–LICENSE

## ffmpeg

The VideoSplit part of the program uses an external application called ffmpeg <https://www.ffmpeg.org/> to split vide
os to multiple images.

## 7za

No part of the application is using 7za.exe but it is used for the ‘download_models.ps1‘ script.

```
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;
import java.util.HashSet;
import java.util.Set;
import java.util.function.Consumer;

public class Callback extends java.rmi.server.UnicastRemoteObject implements ICal
lback {
  CallbackEvent cbe;
  Callback(CallbackEvent cbe) throws RemoteException {
    super();
    this.cbe = cbe;
  }

  public void NewCallback(String sn, String[] dt) throws RemoteException{
    cbe.broadcast(new EventArgs(){{
      serviceName = sn;
      data = dt;
      ipAddress = "";
    }});
  }
  public void NewCallback(String sn, byte[][] dt) throws RemoteException{
    cbe.broadcast(new EventArgs(){{
      serviceName = sn;
      byteData = dt;
      ipAddress = "";
    }});
  }
  public void NewExceptionCallback(String sn, Exception ex){
    cbe.broadcast(new EventArgs(){{
      serviceName = sn;
      ipAddress = "";
      exception = ex;
    }});
  }
}

class EventArgs {
  String serviceName;
  String ipAddress;
  String[] data;
  byte[][] byteData;
  Exception exception;
}

class CallbackEvent{
  private Set<Consumer<EventArgs>> listeners = new HashSet<>();

  public void addListener(Consumer<EventArgs> listener) {
    listeners.add(listener);
  }

  public void broadcast(EventArgs args) {
    listeners.forEach(x -> x.accept(args));
  }
}
```

```
import java.rmi.Naming;
import java.text.MessageFormat;
import java.util.Properties;

public class CallbackServer {
  ICallback cb;
  Properties p;
  CallbackEvent cbe;

  public CallbackServer(Properties p, CallbackEvent cbe) {
    this.p = p;
    this.cbe = cbe;
  }

  public void start() {
    try {
      // Startup the RMI callback server
      cb = new Callback(this.cbe);
      System.out.println("Server Ready");
      System.setProperty("java.rmi.server.hostname", p.getProperty("rmi_registry_host"));
      Naming.rebind(MessageFormat.format("rmi://{0}:{1}/Callback", p.getProperty("rmi_re
gistry_host"),
          p.getProperty("rmi_registry_port")), cb);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Properties;

public class Client {
  static Registry r;
  static TrackerHandler trackerHandler;
  static CallbackEvent cbe;
  static Properties p;

  public static void main(String[] args) {
    p = new Properties();
    File file = new File("client.config.properties");
    if (!file.exists()) {
      try {
        System.out.println("Cant find properties file. Writing default.");
        // Load default config
        FileOutputStream out = new FileOutputStream("client.config.properties");
        p.put("trackers", "localhost:1099");
        p.put("rmi_registry_host", "localhost");
        p.put("rmi_registry_port", "1088");
        p.store(out, null);
      } catch (Exception e) {
        System.err.println("Unable to save file client.config.properties");
        e.printStackTrace();
        return;
      }
    } else {
      try {
        // Load config
        FileInputStream in = new FileInputStream("client.config.properties");
        p.load(in);
      } catch (Exception e) {
        System.err.println("Unable to open file client.config.properties");
        e.printStackTrace();
        return;
      }
    }
    try {
      // Parse RMI Registry port
      int port = Integer.parseInt(p.get("rmi_registry_port").toString());
      // Create a local registry for callback so user does not need to
      r = LocateRegistry.createRegistry(port);
    } catch (Exception e) {
      e.printStackTrace();
      return;
    }
    // Setup callback
    cbe = new CallbackEvent();
    new CallbackServer(p, cbe).start();
    // Setup tracker connection
    trackerHandler = new TrackerHandler(p);
    if (!trackerHandler.Init()) {
      System.exit(1);
    }
    // Run job
    new ClientJobVideoAnalytics(trackerHandler, cbe).run();
  }
}
```

```java
import java.awt.Desktop;
import java.io.File;
import java.io.FileOutputStream;
import java.nio.file.Files;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.text.MessageFormat;
import java.util.Arrays;
import java.util.concurrent.TimeUnit;

public class ClientJobVideoAnalytics {
  TrackerHandler trackerHandler;
  CallbackEvent cbe;
  long startTime;

  public ClientJobVideoAnalytics(TrackerHandler trackerHandler, CallbackEvent cb
e) {
    this.trackerHandler = trackerHandler;
    this.cbe = cbe;
  }

  public void run() {
    // Register callback event
    cbe.addListener((x) -> {
      callBackFromVideoAnalytics(x);
    });
    // Record down start time
    startTime = System.currentTimeMillis();
    // Query for VideoAnalytics Service from tracker
    IServiceNode[] sn = trackerHandler.Query("VideoAnalytics");
    if (sn == null) {
      // If cant find any
      System.err.println("Cant find any service with name {VideoAnalytics} quit.");
      System.exit(1);
    }
    IServiceInterface service = null;
    for (IServiceNode s : sn) {
      try {
        // try and connect to server registry
        Registry cr = LocateRegistry.getRegistry(s.getIp(), s.getPort());
        service = (IServiceInterface) cr.lookup("Service");
        // Found 1 means no exception, enough break out.
        break;
      } catch (Exception e) {
        // Cant connect to that continue look for antoher server
        e.printStackTrace();
        System.out.println(MessageFormat.format("Cant get service with ip {0}. Next.", s.ge
tIp()));
      }
    }
    // No server found
    if (service == null) {
      System.err.println("No service found! Quit.");
      System.exit(1);
    }
    try {
      // Open Video.mp4 file
      File f = new File("Video.mp4");
      // Read as bytes
      byte[] fileContent = Files.readAllBytes(f.toPath());
      // Call VideoAnalytics Service via the found server. This function is
```

```java
      // asynchronous.
      service.RunServiceAsync("VideoAnalytics", new byte[][] { fileContent }, new S
tring[] { "Video.mp4" },
          Client.p.getProperty("rmi_registry_port"));
    } catch (Exception e) {
      // Server cant do it
      System.err.println("Unable to submit job. Quit.");
      e.printStackTrace();
      System.exit(1);
    }
    // Server cant do it
    System.out.println("Job submitted. Waiting for callback.");
  }

  public void callBackFromVideoAnalytics(EventArgs args) {
    // A callback from the server with serviceName as VideoAnalytics
    if (args.serviceName.equals("VideoAnalytics")) {
      // If it is an exception, print it
      if (args.exception != null) {
        System.err.println("Error Callback from VideoAnalytics Service from " + args.ipAddress
);
        args.exception.printStackTrace();
        System.exit(1);
      }
      // Else is result
      System.out.println("Callback from VideoAnalytics service from " + args.ipAddress);
      try {
        // Save the return byte[] as image file.
        File imgFile = new File("graph.png");
        try (FileOutputStream fos = new FileOutputStream(imgFile)) {
          fos.write(args.byteData[0]);
        }
        // Open the image file.
        Desktop.getDesktop().open(imgFile);
        // Print how long it takes
        long stopTime = System.currentTimeMillis();
        long elapsedTime = stopTime - startTime;
        System.out.println("It took: " + elapsedTime + "ms");
        long minutes = TimeUnit.MILLISECONDS.toMinutes(elapsedTime);
        System.out.println("AKA: " + minutes + " minutes.");
        System.exit(0);
      } catch (Exception e) {
        e.printStackTrace();
        System.exit(1);
      }
    }
  }
}
```

```
import java.rmi.RemoteException;

public interface ICallback extends java.rmi.Remote {
  public void NewCallback(String serviceName, String[] data) throws RemoteExcept
ion;
  public void NewCallback(String serviceName, byte[][] data) throws RemoteExcept
ion;
  public void NewExceptionCallback(String serviceName, Exception exception) thro
ws RemoteException;
}
```

```
import java.rmi.RemoteException;

public interface IServiceInterface extends java.rmi.Remote {
  public String[] RunService(String service, String[] data) throws RemoteExcepti
on;
  public boolean RunServiceAsync(String service, String[] data, String callbackP
ort) throws RemoteException;
  public byte[][] RunService(String service, byte[][] data, String[] data2) thro
ws RemoteException;
  public boolean RunServiceAsync(String service, byte[][] data, String[] data2,
String callbackPort) throws RemoteException;
}
```

```java
import java.io.Serializable;

public interface IServiceNode extends Serializable {
  String getIp();

  int getPort();

  String[] getService();
}
```

```
public class ServiceNode implements IServiceNode {
  public String Ip;
  public int Port;
  public String[] Services;

  public String getIp(){
    return Ip;
  }
  public int getPort(){
    return Port;
  }
  public String[] getService(){
    return Services;
  }

  @Override
  public int hashCode() {
    int hc = 0;
    for (String s : Services) {
      hc += s.hashCode();
    }
    return Ip.hashCode() + hc;
  }
}
```

```java
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Properties;

public class TrackerHandler {
  Properties p;
  ArrayList<TrackerInterface> trackers;

  public TrackerHandler(Properties p) {
    this.p = p;
  }

  public Boolean Init() {
    // Get all trackers from properties
    String[] trackerHosts = p.get("trackers").toString().split(",");
    trackers = new ArrayList<>();
    // Loop through all tracker and find all tracker that is available
    for (int i = 0; i < trackerHosts.length; i++) {
      String tracker = trackerHosts[i];
      try {
        TrackerInterface ti = (TrackerInterface) Naming
            .lookup(MessageFormat.format("rmi://{0}/TrackerService", tracker));
        System.out.println(MessageFormat.format("Connected to tracker: {0}", tracker));
        trackers.add(ti);
      } catch (NotBoundException | RemoteException e) {
        e.printStackTrace();
        System.err.println(MessageFormat.format("Unable to connect to tracker with host: {0}."
, tracker));
      } catch (MalformedURLException e) {
        e.printStackTrace();
        System.err.println(MessageFormat.format("The tracker host configured is not valid: {0}.
", tracker));
      }
    }
    if (trackers.size() == 0) {
      System.err.println("No tracker is connected. Unable to continue.");
      return false;
    }
    return true;
  }

  public IServiceNode[] Query(String service) {
    // Loop through all tracker and find one that can return the wanted service
    for (TrackerInterface tracker : trackers) {
      try {
        IServiceNode[] serviceNodes = (IServiceNode[])tracker.GetMeService(servi
ce);
        if (serviceNodes.length != 0) {
          return serviceNodes;
        }
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
    return null;
  }
}
```

```
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;

public interface TrackerInterface extends java.rmi.Remote {
  public Boolean RegisterMeAsService(String[] services, int port, String ipAddr)
 throws RemoteException;

  public IServiceNode[] GetMeService(String service) throws RemoteException;

  public IServiceNode[] GetMeServices(String[] services) throws RemoteException;

  public Boolean RemoveMe(String ipAddr) throws RemoteException, ServerNotActive
Exception;
}
```

```
mkdir bin
javac -d bin *.java
if %errorlevel% neq 0 exit /b %errorlevel%
copy manifest.txt bin
cd bin
jar cvfm client.jar manifest.txt *.class
cd ../
```

```
mkdir -p bin
javac -d bin *.java
cp manifest.txt bin
cd bin
jar cvfm client.jar manifest.txt *.class
cd ../
```

```
Manifest-Version: 1.0
Main-Class: Client
```

```java
import java.io.Serializable;

public interface IServiceNode extends Serializable {
  String getIp();

  int getPort();

  String[] getService();
}
```

```java
public class ServiceNode implements IServiceNode {
  public String Ip;
  public int Port;
  public String[] Services;

  public String getIp(){
    return Ip;
  }
  public int getPort(){
    return Port;
  }
  public String[] getService(){
    return Services;
  }

  @Override
  public int hashCode() {
    int hc = 0;
    for (String s : Services) {
      hc += s.hashCode();
    }
    return Ip.hashCode() + hc;
  }
}
```

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Properties;

public class Tracker {
  static Registry r;
  public static void main(String[] args) {
    Properties p = new Properties();
    // Load the config
    File file = new File("tracker.config.properties");
    if (!file.exists()) {
      try {
        System.out.println("Cant find properties file. Writing default.");
        FileOutputStream out = new FileOutputStream("tracker.config.properties");
        p.put("rmi_registry_port", "1099");
        p.store(out, null);
      } catch (Exception e) {
        System.err.println("Unable to save file tracker.config.properties");
        e.printStackTrace();
        return;
      }
    } else {
      try {
        // If cant load config
        FileInputStream in = new FileInputStream("tracker.config.properties");
        p.load(in);
      } catch (Exception e) {
        System.err.println("Unable to open file tracker.config.properties");
        e.printStackTrace();
        return;
      }
    }
    try {
      // Parse RMI Registry port
      int port = Integer.parseInt(p.get("rmi_registry_port").toString());
      // Create a local registry so user does not need to
      r = LocateRegistry.createRegistry(port);
    } catch (Exception e) {
      e.printStackTrace();
      return;
    }
    // Start tracker server
    new TrackerServer(p).start();
  }
}
```

```
import java.rmi.RemoteException;
import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedHashSet;
import java.util.Map;
import java.util.Set;
import java.rmi.server.ServerNotActiveException;

class TrackerImpl extends java.rmi.server.UnicastRemoteObject implements TrackerI
nterface {
  private Map<String, ArrayList<IServiceNode>> serviceDict;
  private ArrayList<IServiceNode> serviceNodes;

  TrackerImpl() throws RemoteException {
    super();
    serviceDict = new HashMap<String, ArrayList<IServiceNode>>();
    serviceNodes = new ArrayList<>();
  }

  private void Log(String log) {
    System.out.println(log);
  }
  // Register a server as a service
  public Boolean RegisterMeAsService(String[] services, int port, String ipAddr)
 throws RemoteException {
    Log(MessageFormat.format("Request: {0}", ipAddr));
    // Loop trough all service current service node. If the ip already registere
d.
    for (IServiceNode sn : serviceNodes) {
      if (sn.getIp().equals(ipAddr)) {
        Log(MessageFormat.format("IP: {0} already added.", ipAddr));
        return false;
      }
    }
    // Create a new service node
    ServiceNode serviceNode = new ServiceNode();
    serviceNode.Ip = ipAddr;
    serviceNode.Services = services;
    serviceNode.Port = port;
    // Make Map
    for (String s : services) {
      ArrayList<IServiceNode> sn = serviceDict.get(s);
      if (sn == null) {
        sn = new ArrayList<IServiceNode>();
        serviceDict.put(s, sn);
      }
      sn.add(serviceNode);
    }
    Log(MessageFormat.format("IP: {0}:{1} added with services {2}", ipAddr, port, Arrays.t
oString(services)));
    serviceNodes.add(serviceNode);
    return true;
  }
  // Return all serviceNodes that matches the service requested
  public IServiceNode[] GetMeService(String service) throws RemoteException{
    ArrayList<IServiceNode> sn = serviceDict.get(service);
    if(sn == null) return new IServiceNode[0];
    return sn.toArray(new IServiceNode[sn.size()]);
  }
```

```
  // Return all serviceNodes that matches the service requested
  public IServiceNode[] GetMeServices(String[] services) throws RemoteException
{
    ArrayList<IServiceNode> sns = new ArrayList<>();
    for (String s : services) {
      ArrayList<IServiceNode> sn = serviceDict.get(s);
      if (sn != null) {
        sns.addAll(sn);
      }
    }
    return MergeServiceNodes(sns);
  }

  // Delist the serviceNode that matches the ipAddress
  public Boolean RemoveMe(String ipAddr) throws RemoteException, ServerNotActive
Exception {
    IServiceNode removeNode = null;
    for (IServiceNode sn : serviceNodes) {
      if (sn.getIp().equals(ipAddr)) {
        removeNode = sn;
        break;
      }
    }
    if(removeNode == null){
      return false;
    }
    Log(MessageFormat.format("IP: {0} removed", ipAddr));
    for (String service : removeNode.getService()) {
      serviceDict.get(service).remove(removeNode);
    }
    serviceNodes.remove(removeNode);
    return true;
  }

  private IServiceNode[] MergeServiceNodes(ArrayList<IServiceNode> services) {
    Set<IServiceNode> set = new LinkedHashSet<>();
    set.addAll(services);
    services.clear();
    services.addAll(set);
    return services.toArray(new IServiceNode[services.size()]);
  }
}
```

```java
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;

public interface TrackerInterface extends java.rmi.Remote {
  public Boolean RegisterMeAsService(String[] services, int port, String ipAddr)
 throws RemoteException;

  public IServiceNode[] GetMeService(String service) throws RemoteException;

  public IServiceNode[] GetMeServices(String[] services) throws RemoteException;

  public Boolean RemoveMe(String ipAddr) throws RemoteException, ServerNotActive
Exception;
}
```

```
import java.rmi.Naming;
import java.rmi.server.UnicastRemoteObject;
import java.text.MessageFormat;
import java.util.Properties;

public class TrackerServer {
  TrackerImpl t;
  Properties p;
  public TrackerServer(Properties p){
    this.p = p;
  }
  public void start() {
    try {
      // Register tracker instance
      t = new TrackerImpl();
      System.out.println("Server Ready");
      Naming.rebind(MessageFormat.format("rmi://localhost:{0}/TrackerService", p.get("rmi_re
gistry_port")), t);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
mkdir bin
javac -d bin *.java
if %errorlevel% neq 0 exit /b %errorlevel%
copy manifest.txt bin
cd bin
jar cvfm tracker.jar manifest.txt *.class
cd ../
```

```
mkdir -p bin
javac -d bin *.java
cp manifest.txt bin
cd bin
jar cvfm tracker.jar manifest.txt *.class
cd ../
```

```
Manifest-Version: 1.0
Main-Class: Tracker
```

```java
import java.rmi.RemoteException;

public class CallBackHandler{
  private ICallback callback;
  CallBackHandler(ICallback callback){
    this.callback = callback;
  }
  public boolean ExecuteCallback(String serviceName, String[] data){
    try {
      callback.NewCallback(serviceName, data);
    } catch (RemoteException e) {
      e.printStackTrace();
      return false;
    }
    return true;
  }
  public boolean ExecuteCallback(String serviceName, byte[][] data){
    try {
      callback.NewCallback(serviceName, data);
    } catch (RemoteException e) {
      e.printStackTrace();
      return false;
    }
    return true;
  }
  public boolean ExecuteExceptionCallback(String serviceName, Exception e){
    try {
      callback.NewExceptionCallback(serviceName, e);
    } catch (RemoteException ex) {
      ex.printStackTrace();
      return false;
    }
    return true;
  }
}
```

```
import java.rmi.RemoteException;

public interface ICallback extends java.rmi.Remote {
  public void NewCallback(String serviceName, String[] data) throws RemoteExcept
ion;
  public void NewCallback(String serviceName, byte[][] data) throws RemoteExcept
ion;
  public void NewExceptionCallback(String serviceName, Exception exception) thro
ws RemoteException;
}
```

```java
public interface IService{
  String[] run(String[] data) throws Exception;
  byte[][] run(byte[][] data, String[] data2) throws Exception;
}
```

```
import java.rmi.RemoteException;

public interface IServiceInterface extends java.rmi.Remote {
  public String[] RunService(String service, String[] data) throws RemoteExcepti
on;
  public boolean RunServiceAsync(String service, String[] data, String callbackP
ort) throws RemoteException;
  public byte[][] RunService(String service, byte[][] data, String[] data2) thro
ws RemoteException;
  public boolean RunServiceAsync(String service, byte[][] data, String[] data2,
String callbackPort) throws RemoteException;
}
```

```
import java.io.Serializable;

public interface IServiceNode extends Serializable {
  String getIp();

  int getPort();

  String[] getService();
}
```

```java
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.Properties;

public class Server {
  static Registry r;
  static Properties p;
  public static void main(String[] args) {
   Properties p = new Properties();
   Server.p = p;
   // Load the config
   File file = new File("server.config.properties");
   if (!file.exists()) {
     try {
       System.out.println("Cant find properties file. Writing default.");
       FileOutputStream out = new FileOutputStream("server.config.properties");
       p.put("tracker", "localhost:1099");
       p.put("services", "VideoAnalytics,VideoSplit,ImageAnalytics,ImageAnalyticsGraph");
       p.put("rmi_registry_host", "localhost");
       p.put("rmi_registry_port", "1000");
       p.put("image_analytics_model_dir", "models/ssd_inception_v2_coco_2017_11_17/saved_model");
       p.put("image_analytics_label", "labels/mscoco_label_map.pbtxt");
       p.put("image_analytics_simulate", "0");
       p.put("ffmpeg_command", "./ffmpeg");
       p.store(out, null);
     } catch (Exception e) {
       System.err.println("Unable to save file server.config.properties");
       e.printStackTrace();
       return;
     }
   } else {
     try {
      // If cant load config
       FileInputStream in = new FileInputStream("server.config.properties");
       p.load(in);
     } catch (Exception e) {
       System.err.println("Unable to open file server.config.properties");
       e.printStackTrace();
       return;
     }
   }
   try {
     // Set hostname of registry to {hostname} or not it will use first network i
nterface and cause problems
     System.setProperty("java.rmi.server.hostname",p.getProperty("rmi_registry_host"));
     // Parse RMI Registry port
     int port = Integer.parseInt(p.get("rmi_registry_port").toString());
     // Create a local registry so user does not need to
     r = LocateRegistry.createRegistry(port);
   } catch (Exception e) {
     e.printStackTrace();
     return;
   }
   // Start server
   new ServerService(p).run();
  }
}
```

```java
import java.rmi.Naming;
import java.text.MessageFormat;
import java.util.Map;
import java.util.Properties;

public class ServerService {
  class ShutdownThread extends Thread{
    public TrackerInterface t;
    public Properties properties;
    @Override
    public void run() {
      // Function that delist itself from tracker
      try {
        System.out.println("Removing myself from tracker.");
        t.RemoveMe(properties.getProperty("rmi_registry_host"));
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
  Properties properties;
  String[] services;
  public static TrackerInterface t;

  ServerService(Properties p) {
    properties = p;
    p.list(System.out);
    services = p.get("services").toString().split(",");
  }

  void run() {
    try {
      // Create a new shutdown thread
      ShutdownThread st = new ShutdownThread();
      st.properties = properties;
      // Add shutdown thread to hook incase user use ctrl+c to stop the app
      Runtime.getRuntime().addShutdownHook(st);
      // Look for the tracker
      t = (TrackerInterface) Naming.lookup(MessageFormat.format("rmi://{0}/TrackerServ
ice", properties.get("tracker")));
      st.t = t;
      // Advertise itself to the tracker
      t.RegisterMeAsService(services, Integer.parseInt(properties.get("rmi_registry
_port").toString()), properties.getProperty("rmi_registry_host"));
      System.out.println("Connected to tracker");
      IServiceInterface service = new ServiceImpl();
      // Create new service instance and bind it
      Naming.rebind(MessageFormat.format("rmi://{0}/Service", properties.getProperty(
"rmi_registry_host") + ":" + properties.getProperty("rmi_registry_port")), service);
      System.out.println("Service Published");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.Arrays;

public class ServiceImageAnalytics implements IService {
  public String[] run(String[] data) throws Exception {
    throw new Exception("This is not implemented. Please use byte[][] variant.");
  }
  // data[0] Array of bytes
  // data[0][0] bytes of Images
  // data2 not used
  public byte[][] run(byte[][] data, String[] data2) throws Exception {
    // If Simulate is on, use hard coded value
    if(Server.p.getProperty("image_analytics_simulate").equals("0")){
      return ExecuteImageAnalytics(data);
    }
    return ExecuteImageAnalyticsSimulate();
  }
  // Execute tensor flow detect object
  private byte[][] ExecuteImageAnalytics(byte[][] data) throws Exception{
    // Generate CSV file
    File csvFile = new File("out.csv");
    ArrayList<String> cmdA = new ArrayList<>();
    cmdA.add("java");
    cmdA.add("-jar");
    cmdA.add("detect-object.jar");
    cmdA.add(Server.p.getProperty("image_analytics_model_dir"));
    cmdA.add(Server.p.getProperty("image_analytics_label"));
    cmdA.add(csvFile.toString());
    try {
      // Write the image to disk
      File basePath = new File("image_analytics");
      basePath.mkdir();
      File dir = Files.createTempDirectory(basePath.toPath(), "image_analytics").toF
ile();
      for (int i = 0; i < data.length; i++) {
        File f = new File(dir, i + ".png");
        byte[] b = data[i];
        try (FileOutputStream fos = new FileOutputStream(f)) {
          fos.write(b);
        }
      // Add to command
        cmdA.add(f.toString());
      }
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
    }
    try {
      String[] cmd = new String[cmdA.size()];
      cmdA.toArray(cmd);
      System.out.println(Arrays.toString(cmd));
      // Call the tensor flow jar file to process images
      ProcessBuilder ps = new ProcessBuilder(cmd);
      ps.redirectErrorStream(true);
      Process pr = ps.start();
```

```
      // Read the output stream
      BufferedReader in = new BufferedReader(new InputStreamReader(pr.getInputSt
ream()));
      String line;
      while ((line = in.readLine()) != null) {
          System.out.println(line);
      }
      in.close();
      pr.waitFor();
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
    }
    try {
      // Read the csv file and return it
      return new byte[][] { Files.readAllBytes(csvFile.toPath()) };
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
    }
  }
  private byte[][] ExecuteImageAnalyticsSimulate(){
    return new byte[][]{"cat,15\ndog,11\nturtle,22\nbob,100\norrange,3".getBytes()};
  }
}
```

```java
import org.knowm.xchart.BitmapEncoder;
import org.knowm.xchart.PieChart;
import org.knowm.xchart.PieChartBuilder;

import java.io.IOException;

public class ServiceImageAnalyticsGraph implements IService {
  public String[] run(String[] data) throws Exception {
    throw new Exception("This is not implemented. Please use byte[][] variant.");
  }

  // data[0] = csv content
  // data2 not used
  public byte[][] run(byte[][] data, String[] data2) throws Exception {
    byte[] csvStringByte = data[0];
    String csvString = new String(csvStringByte);
    // split the csv
    String[] splitNewLineCsv = csvString.split("\n");
    // New piechart builder
    PieChart chart = new PieChartBuilder().width(800).height(600).title("Pie Chart
").build();
    chart.getStyler().setCircular(false);
    // Add the items
    for (String s : splitNewLineCsv) {
      String[] splitStr = s.split(",");
      chart.addSeries(splitStr[0], Integer.parseInt(splitStr[1]));
    }
    // Convert the image bitmap to byte
    byte[] imageData;
    try {
      imageData = BitmapEncoder.getBitmapBytes(chart, BitmapEncoder.BitmapFormat
.PNG);
    } catch (IOException e) {
      e.printStackTrace();
      throw e;
    }
    // return it
    return new byte[][] { imageData };
  }
}
```

```java
import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.text.MessageFormat;

public class ServiceImpl extends java.rmi.server.UnicastRemoteObject implements I
ServiceInterface {
  private CallBackHandler callbackHandler;

  ServiceImpl() throws RemoteException {
    super();
  }

  // Get the service to run via Reflection
  private IService getServiceClass(String serviceName)
      throws ClassNotFoundException, InstantiationException, IllegalAccessExcept
ion {
    Class serviceClass = Class.forName("Service" + serviceName);
    return (IService) serviceClass.newInstance();
  }

  // Register client callback
  private void RegisterCallback(String host, String clientPort)
      throws RemoteException, NotBoundException, MalformedURLException {
    if (this.callbackHandler != null)
      return;
    ICallback callback = (ICallback) Naming.lookup(MessageFormat.format("rmi://{0}:
{1}/Callback", host, clientPort));
    this.callbackHandler = new CallBackHandler(callback);
  }

  // Run the service (String variant). This function is synchronous.
  public String[] RunService(String service, String[] data) throws RemoteExcepti
on {
    try {
      return getServiceClass(service).run(data);
    } catch (Exception e) {
      e.printStackTrace();
      throw new RemoteException("Unable to run service", e);
    }
  }

  // Run the service (String variant). This function is asynchronous.
  public boolean RunServiceAsync(String service, String[] data, String callbackP
ort) throws RemoteException {
    try {
      // Get connecting client IP
      String clientHost = getClientHost();
      // Create new callback
      RegisterCallback(clientHost, callbackPort);
    } catch (Exception e) {
      e.printStackTrace();
      throw new RemoteException("Failed to register callback. Unable to get host", e);
    }
    // Run execution in another thread so that the client does not need to wait
    Thread t = new Thread(() -> {
      try {
        // Run Service
        String[] result = RunService(service, data);
        // Call client callback when execution finished
        callbackHandler.ExecuteCallback(service, result);
```

```java
      } catch (Exception e) {
        e.printStackTrace();
        // Got an exception, tell client got exception via callback
        callbackHandler.ExecuteExceptionCallback(service, e);
      }
    });
    t.start();
    return true;
  }

  // Run the service (Byte[][] variant). This function is synchronous.
  public byte[][] RunService(String service, byte[][] data, String[] data2) thro
ws RemoteException {
    try {
      return getServiceClass(service).run(data, data2);
    } catch (Exception e) {
      e.printStackTrace();
      throw new RemoteException("Unable to run service", e);
    }
  }

  // Run the service (Byte[][] variant). This function is asynchronous.
  public boolean RunServiceAsync(String service, byte[][] data, String[] data2,
String callbackPort)
      throws RemoteException {
    try {
      // Get connecting client IP
      String clientHost = getClientHost();
      // Create new callback
      RegisterCallback(clientHost, callbackPort);
    } catch (Exception e) {
      e.printStackTrace();
      throw new RemoteException("Failed to register callback. Unable to get host", e);
    }
    // Run execution in another thread so that the client does not need to wait
    Thread t = new Thread(() -> {
      try {
        // Run Service
        byte[][] result = RunService(service, data, data2);
        // Call client callback when execution finished
        callbackHandler.ExecuteCallback(service, result);
      } catch (Exception e) {
        e.printStackTrace();
        // Got an exception, tell client got exception via callback
        callbackHandler.ExecuteExceptionCallback(service, e);
      }
    });
    t.start();
    return true;
  }
}
```

```java
public class ServiceNode implements IServiceNode {
  public String Ip;
  public int Port;
  public String[] Services;

  public String getIp(){
    return Ip;
  }
  public int getPort(){
    return Port;
  }
  public String[] getService(){
    return Services;
  }

  @Override
  public int hashCode() {
    int hc = 0;
    for (String s : Services) {
      hc += s.hashCode();
    }
    return Ip.hashCode() + hc;
  }
}
```

```
import java.rmi.Naming;
import java.text.MessageFormat;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

class ServiceVideoAnalytics implements IService {
  public String[] run(String[] data) throws Exception {
    throw new Exception("This is not implemented. Please use byte[][] variant.");
  }

  // data[0] = video file
  // data2[0] = filename
  public byte[][] run(byte[][] data, String[] data2) throws Exception {
    // Setup all service nodes first before executing
    ArrayList<IServiceInterface> videoSplitNode = GetServices("VideoSplit", 1);
    ArrayList<IServiceInterface> imageAnalyticsNode = GetServices("ImageAnalytics",
 4);
    ArrayList<IServiceInterface> imageAnalyticsGraphNode = GetServices("ImageAnal
yticsGraph", 1);
    // If one of the service have no node, tell throw error
    if (videoSplitNode == null || imageAnalyticsNode == null || imageAnalyticsGr
aphNode == null) {
      System.err.println("Unable to find one of the node.");
      throw new Exception("Unable to find one of the node.");
    }
    byte[][] fileBytes = GenerateVideoThumb(data[0], data2[0], videoSplitNode.ge
t(0));
    byte[] nameToOccurance = ImageAnalytics(fileBytes, imageAnalyticsNode);
    byte[] graphImg = GraphIt(nameToOccurance, imageAnalyticsGraphNode.get(0));
    return new byte[][]{graphImg};
  }

  // Call the split video service node
  private byte[][] GenerateVideoThumb(byte[] data, String filename, IServiceInte
rface service) throws Exception {
    byte[][] fileBytes;
    try {
      fileBytes = service.RunService("VideoSplit", new byte[][] { data }, new Stri
ng[] { filename });
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
    }
    return fileBytes;
  }

  // Split the task to (max of 4) nodes
  private byte[] ImageAnalytics(byte[][] files, ArrayList<IServiceInterface> ser
vices) throws Exception {
    ArrayList<byte[]>[] splitedFiles = SplitFiles(files, services.size());
    Thread[] threads = new Thread[services.size()];
    byte[][] result = new byte[services.size()][];
    for (int i = 0; i < services.size(); i++) {
      final int index = i;
      final ArrayList<byte[]> filesToRun = splitedFiles[i];
      final IServiceInterface serviceToRunAt = services.get(i);
      // Create threads for each service to call. so can simultaneous calls
      Thread t = new Thread(() -> {
        byte[][] bts = new byte[filesToRun.size()][];
        for (int y = 0; y < filesToRun.size(); y++) {
          try {
```

```
            bts[y] = filesToRun.get(y);
          } catch (Exception e) {
            e.printStackTrace();
          }
        }
        try {
          System.out.println("Running service with T: " + Thread.currentThread().getId(
));
          byte[][] r = serviceToRunAt.RunService("ImageAnalytics", bts, null);
          result[index] = r[0];
        } catch (Exception e) {
          e.printStackTrace();
        }
      });
      t.start();
      threads[i] = t;
    }
    // Wait for all nodes to finish
    for (int i = 0; i < threads.length; i++) {
      try {
        threads[i].join();
      } catch (Exception e) {
        e.printStackTrace();
        throw e;
      }
    }
    // Combine the result of all nodes
    StringBuilder sb = new StringBuilder();
    Map<String, Integer> nameToOccurance = new HashMap<>();
    for (int i = 0; i < result.length; i++) {
      byte[] csvB = result[i];
      if(csvB == null){
        continue;
      }
      String csvContent = new String(csvB);
      csvContent = csvContent.replace("\r", "");
      String[] newLineSplit = csvContent.split("\n");
      for (String s : newLineSplit) {
        if (!s.equals("")) {
          String name = s.split(",")[0];
          if (nameToOccurance.get(name) == null) {
            nameToOccurance.put(name, 0);
          }
          nameToOccurance.put(name, nameToOccurance.get(name) + 1);
        }
      }
    }
    if(nameToOccurance.keySet().size() == 0){
      throw new Exception("No result from Image Analytics.");
    }
    for (String key : nameToOccurance.keySet()) {
      sb.append(key).append(",").append(nameToOccurance.get(key)).append("\n");
    }
    System.out.println("CSV Return: " + sb.toString());
    return sb.toString().getBytes();
  }

  // Call the Graph node
  private byte[] GraphIt(byte[] nameToOccuranceCsv, IServiceInterface service) t
hrows Exception{
    byte[][] fileBytes;
    try {
      fileBytes = service.RunService("ImageAnalyticsGraph", new byte[][] { nameToOcc
uranceCsv }, null);
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
    }
    return fileBytes[0];
  }

  // Split files to nodes
  private ArrayList<byte[]>[] SplitFiles(byte[][] files, int parts) {
    ArrayList<byte[]>[] splitedFiles = new ArrayList[parts];
    for (int i = 0; i < parts; i++) {
      splitedFiles[i] = new ArrayList<byte[]>();
    }
    for (int i = 0; i < files.length; i++) {
      splitedFiles[i % parts].add(files[i]);
    }
    return splitedFiles;
  }

  private ArrayList<IServiceInterface> GetServices(String serviceName, int max)
{
    ArrayList<IServiceInterface> services = new ArrayList<>();
    try {
      IServiceNode[] sn = (IServiceNode[])ServerService.t.GetMeService(serviceNa
me);
      for (IServiceNode s : sn) {
        try {
          services.add((IServiceInterface) Naming
              .lookup(MessageFormat.format("rmi://{0}/Service", s.getIp() + ":" + s.g
etPort())));
          if (services.size() == max) {
            break;
          }
        } catch (Exception e) {
          e.printStackTrace();
          System.out.println(MessageFormat.format("Cant get service with ip {0}. Next.", s.
getIp()));
        }
      }
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    }
    if (services.size() == 0) {
      System.err.println("Unable to fine any node named " + serviceName);
      return null;
    }
    return services;
  }
}
```

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.nio.file.Files;
import java.util.Arrays;

class ServiceVideoSplit implements IService {
  public String[] run(String[] data) throws Exception {
    throw new Exception("This is not implemented. Please use byte[][] variant.");
  }

  // data[0] = video file
  // data2[0] = filename
  public byte[][] run(byte[][] data, String[] data2) throws Exception {
    try {
      // Save the video file
      new File("temp/").mkdirs();
      new File("temp/" + data2[0] + "out/").mkdirs();
      try (FileOutputStream fos = new FileOutputStream("temp/" + data2[0])) {
        fos.write(data[0]);
      }
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
    }
    // split file
    try {
      // Setup cmd
      String[] cmd = { Server.p.getProperty("ffmpeg_command"), "-i", "temp/" + data
2[0], "-vf", "scale=720:-1,fps=4",
          "temp/" + data2[0] + "out/out%d.png" };
      System.out.println(Arrays.toString(cmd));
      ProcessBuilder ps = new ProcessBuilder(cmd);
      ps.redirectErrorStream(true);
      // Call ffmpeg to split video 4 frames per seconds
      Process pr = ps.start();
      // Read the output stream
      BufferedReader in = new BufferedReader(new InputStreamReader(pr.getInputSt
ream()));
      String line;
      while ((line = in.readLine()) != null) {
        System.out.println(line);
      }
      in.close();
      pr.waitFor();
      if (pr.exitValue() != 0)
        throw new Exception("ffmpeg return non 0 exit code.");
    } catch (Exception e) {
      e.printStackTrace();
      throw e;
    }
    // Read the output image files as byte
    File dir = new File("temp/" + data2[0] + "out/");
    File[] files = dir.listFiles();
    byte[][] bytes = new byte[files.length][];
    for (int i = 0; i < files.length; i++) {
      try {
        bytes[i] = Files.readAllBytes(files[i].toPath());
      } catch (Exception e) {
        e.printStackTrace();
```

```
      }
    }
    return bytes;
  }
}
```

```
import java.rmi.RemoteException;
import java.rmi.server.ServerNotActiveException;

public interface TrackerInterface extends java.rmi.Remote {
  public Boolean RegisterMeAsService(String[] services, int port, String ipAddr)
 throws RemoteException;

  public IServiceNode[] GetMeService(String service) throws RemoteException;

  public IServiceNode[] GetMeServices(String[] services) throws RemoteException;

  public Boolean RemoveMe(String ipAddr) throws RemoteException, ServerNotActive
Exception;
}
```

```
mkdir bin
javac -classpath ../../lib/detect-object.jar;../../lib/xchart-3.5.2.jar -d bin *
.java
if %errorlevel% neq 0 exit /b %errorlevel%
copy manifest.txt bin
cd bin
jar cvfm server.jar manifest.txt *.class
cd ../
```

```
mkdir -p bin
javac -classpath '../../lib/detect-object.jar:../../lib/xchart-3.5.2.jar' -d bin *.java
cp manifest.txt bin
cd bin
jar cvfm server.jar manifest.txt *.class
cd ../
```

```
Manifest-Version: 1.0
Main-Class: Server
```

Table of Contents