# Reference Manual - CaSS

**Table of Contents**



### 1. Overview

This document includes information about Camera Surveillance System (known as CaSS). CaSS consists of two parts: servers and clients. The servers are cameras and clients are instances of the graphical user interface used to view and control CaSS.

### 2. User's Guide

The client program is started with the following command:

```
java -jar CaSS_client.jar address1 address2
```

The user operates CaSS using a client program with a graphical user interface seen in Figure 1.

When the program is in idle mode pictures are displayed every 5 seconds. Idle mode can be manually switched on by pressing the "Idle Mode" button.

When the program is in movie mode pictures are displayed more fluidly. Movie mode can be manually switched on by pressing the "Movie Mode" button.

When the program is in movie mode the movie can be played in two different ways: asynchronous playback and synchronous playback. The program *automatically* switches between asynchronous playback and synchronous playback depending on how much of a delay there is between images received from the server. During asynchronous playback the movie frames are displayed as they arrive. During synchronous playback the movie frames are displayed with a delay, staying true to how the movie was recorded by the server.

When the program is in automatic mode the program *automatically* switches between idle mode and movie mode. If any motion is detected by the server, the program switches over to movie mode. If no motion is detected by the server, the program switches over to idle mode.

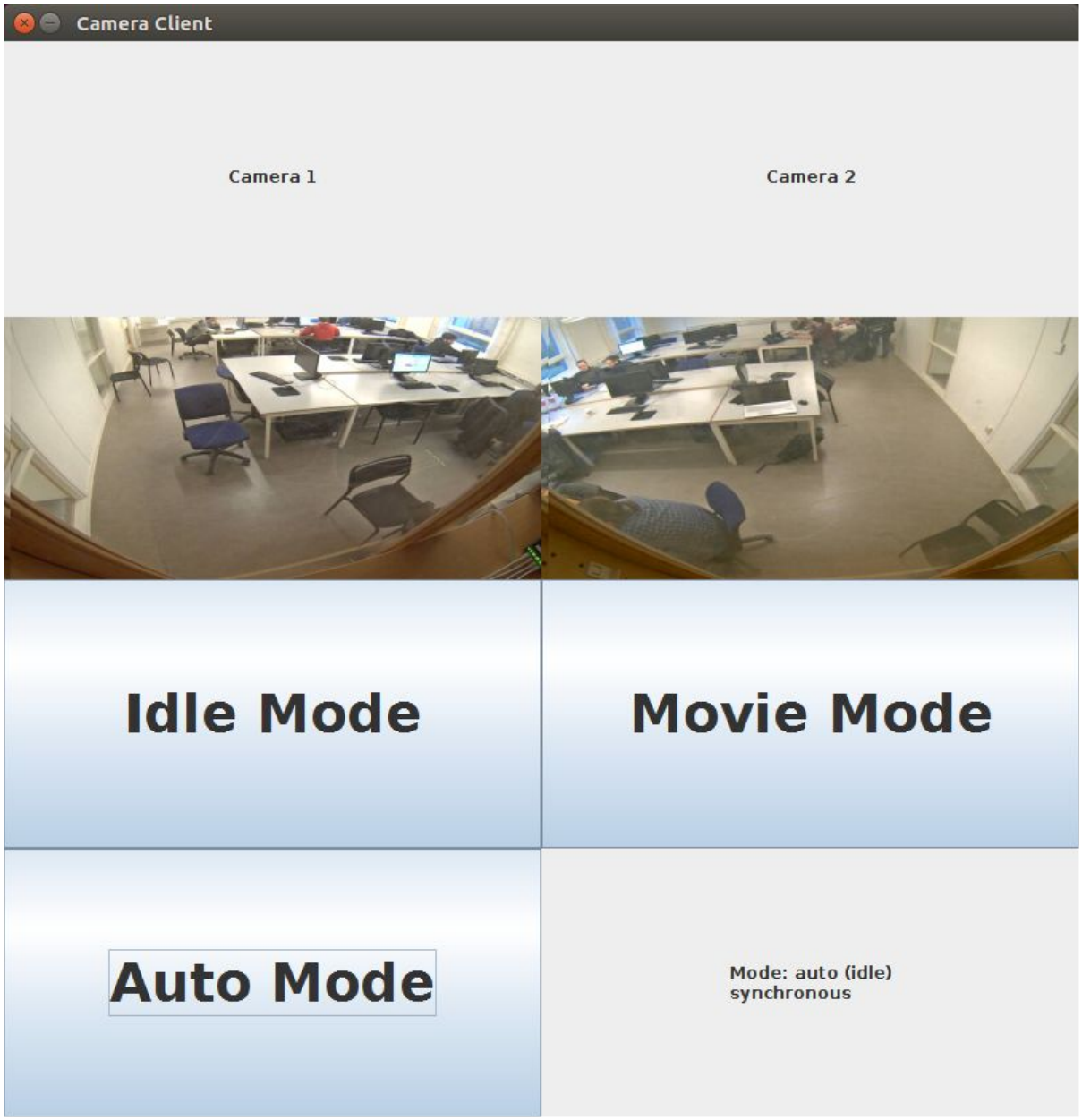


**Figure 1: The graphical user interface**

### 3. Technical Implementation

CaSS is implemented as a client-server system where the camera acts as a server sending images to the client where the images are displayed using a GUI described in the User's Guide.

#### 3.1 Server Implementation

As seen below in Figure 2, the server is minimalistically designed building on a platform using 3 threads operating with a monitor. The *In* and *Out* threads communicate with the client and send image data and receive mode changes. The *Read* thread reads images from the camera hardware.

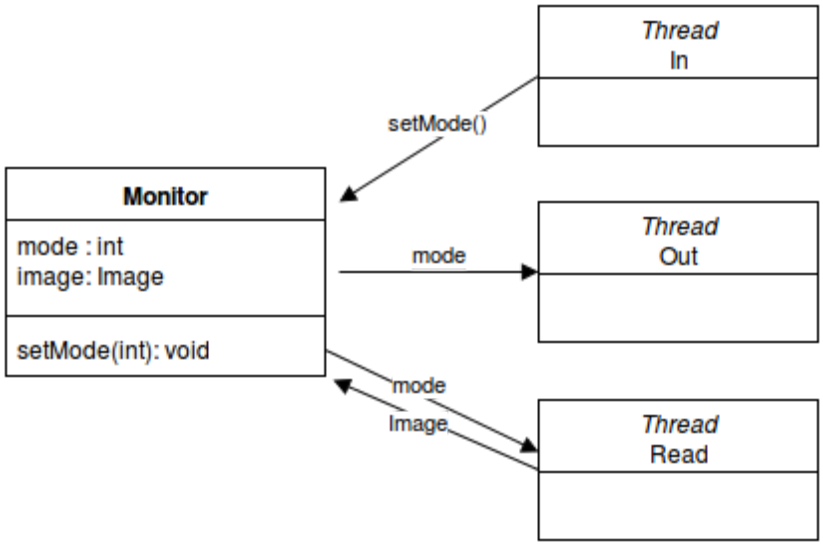


**Figure 2: Server UML**

#### 3.2 Client Implementation

As seen below in Figure 3, the client is implemented using 3 threads and and a GUI communicating through a monitor. The *In* and *Out* threads communicate with the client and can request mode changes and receive image data. The renderer retrieves images and mode changes from the monitor and forwards the rendered images as described in section 3.6.

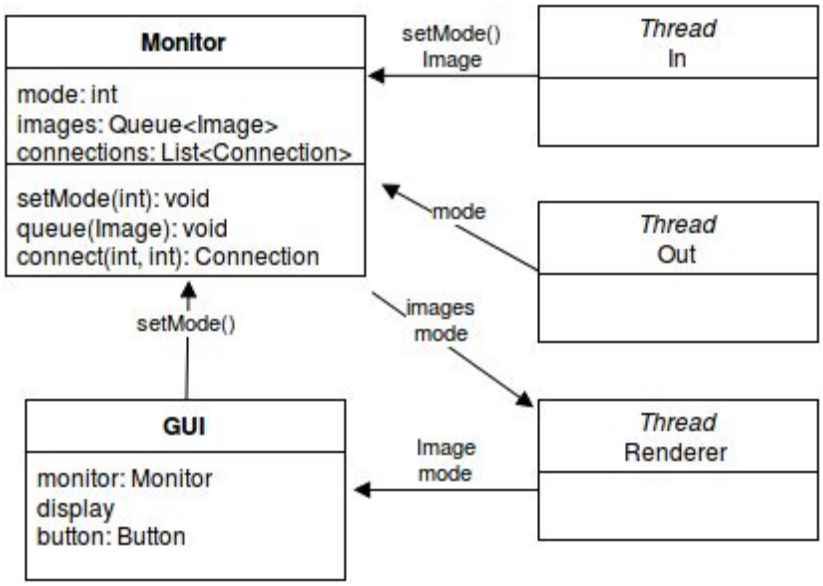


**Figure 3: Client UML**

### 3.3 Connecting the Server and Client

The client and server communicate through sockets.

### 3.4 The Network Protocol

The network protocol is designed according to the following specifications:

**Server → Client**

| Byte size | Description |
|---|---|
| **1** | Packet type |
| **1** | Mode |
| **8** | Timestamp |
| **4** | Data length |
| ***** | Image data |

** = Variable length determined by the **data length** field that is given by the camera.*

**Client → Server**

| Byte size | Description |
|---|---|
| **1** | Packet type |
| **1** | Mode |

**Packet type:** *1* = Server to Client, *2* = Client to Server
**Mode:** *1* = Idle mode, *2* = Movie mode, *3* = Auto mode

### 3.5 Sending the Image-data

The image data is stored in an Image object.

```
public class Image {
     private long timestamp, clientTime;
     private int cameraId;
     private byte[] data;
     private boolean motion;
}
```

The *timestamp* is used to calculate the time difference between two images in the renderer as described in section 3.6. The *cameraId* is used to support multiple cameras. Data from the visual picture captured by the camera is stored in the byte array *data*. The boolean *motion* is used as a signal to the receiver whether or not the camera has detected motion and the images should be viewed in movie mode. Note it is also possible to manually switch between idle and movie mode through the client.

#### 3.6 Rendering data

There are two modes the client can be in: idle mode and movie mode. During idle mode, only one image is rendered every 5 seconds. During movie mode, the images are rendered more frequently (as if it were a video).

There are two ways to render images in movie mode: asynchronous playback and synchronous playback. The client automatically switches over to asynchronous playback if the delay between arriving images exceeds 200 ms, otherwise the client stays in synchronous mode. In order to prevent frequent switching between playback modes, a debouncing mechanism is implemented.

For the curious reader, a simple version of our debouncing mechanism looks like following:

```
long t = System.currentTimeMillis();
if (t - lastPlaybackModeSwitch > 3000L)
     this.switchPlaybackMode();
```

#### 3.6.1 Synchronous playback

When in synchronous playback mode, the images are rendered with different delays depending on the delay between two images captured on the server. This way, the playback is rendered at *normal* speed and stays true to its original recording.

### 3.6.2 Asynchronous playback

When in asynchronous playback mode, the images are rendered as they arrive from the server. This means that we only look at the latest images from the server and ignore having an accurate delay timing between frames. This means that some of the playback may be rendered at speeds faster and slower than normal speed, and therefore does not stay true to its original recording.