Java Client for Player/Stage

"Experience the power of Java"

Maxim A. Batalin (maxim@robotics.usc.edu)

How to…

This brief paper would introduce basic concepts of construction programs using Java Client. Note that Java Client is a client and therefore, the major concepts are the same as those of Player/Stage. Thus, the major assumption is that the reader introduced himself with concepts that are provided in Player/Stage manuals. In general, Java Client is similar to the C++ client with respect to general architecture. Hence, the major construction steps are:

```
1:  Connect to robot by constructing a PlayerClient
    object
2:  Create devices that are to be used in the
    program by requesting them from the PlayerClient
    object.
3:  while(someConditionToFinishIsNotTrue) {
4:    Read the data from devices
5:    Based on received data determine actions
6:  }
```

Therefore, the lines 4-5 would be repeated until a certain condition was met: mission accomplished or loop forever are possible values. For better understanding of presented concepts consider a simple program:

```
 1:  import Javaclient.src.*;
 2:  public class CircleWalk {
 3:    public static void main(String[] args) {
 4:      PlayerClient pc = new PlayerClient("localhost",6665);
 5:      PositionPlayerDevice ppd = pc.requestPosition('a');
 6:      while (true) {
 7:        pc.readAll();
 8:        ppd.setSpeed(100, 30);
 9:      }
10:    }
11:  }
```

At line 1 the *Java Client* library is being imported. At line 4 the *PlayerClient* object is being created. The two parameters that are needed to connect to the robot are the *ServerName* ("localhost" means connect to machine that executes the program) and *PortOfConnection* (6665 – port number for the connection). Line 5 creates a device – *PositionPlayerDevice*, which is being created by corresponding request to *PlayerClient*

object. The only argument of the *request()* method is character describing device access codes (refer to Player/Stage 1.2 manual p. 20, table 5.2). Lines 6-9 describe a "life cycle" of the program. At line 7 a *readAll()* method is being called, which reads data for every created device. Thus, after line 7 user can access new data returned from the server. Line 8 contains command issued to the *PositionPlayerDevice – setSpeed(translationalSpeed, turnrate)*, which constantly advances robot in circular orbit.

1. The *PlayerClass* class

1.1 Constructor

```
public PlayerClient(String serverName, int portNumber);
```

Where *serverName* is the URL of the server on which the program should be run ("localhost" means connect to machine that executes the program) and *portNumber* is the port number of the connection.

1.2 Methods

Request device access - Player/Stage 1.2 manual (further abbreviated as PS) p.20

```
public MiscPlayerDevice requestMisc(char r);
public GripperPlayerDevice requestGripper(char r;
public PositionPlayerDevice requestPosition(char r);
public SonarPlayerDevice requestSonar(char r);
public LaserPlayerDevice requestLaser(char r);
public VisionPlayerDevice requestVision(char r);
public PtzPlayerDevice requestPtz(char r);
public AudioPlayerDevice requestAudio(char r);
public BeaconPlayerDevice requestBeacon(char r);
public BroadcastPlayerDevice requestBroadcast(char r);
public SpeechPlayerDevice requestSpeech(char r);
public GPSPlayerDevice requestGPS(char r);
public TruthPlayerDevice requestTruth(char r);
public BPSPlayerDevice requestBPS(char r);
```

Change data mode – PS (p. 20)

```
public void requestDataMode(byte mode);
```

Request one round of data – PS (p. 21)

```
public void requestOneRoundData();
```

Change data frequency – PS (p.21)

```
public void requestDataFrequency(short frequency);
```

Authentication – PS (p.21)

```
public void requestAuthentication(byte[] key);
```

Devices

2. The *MiscPlayerDevice* class

2.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Methods return exactly the same variable as in PS (p. 22)

```
public byte getFrontBumpers();
public byte getReadBumpers();
public byte getBattery();
public byte getAnalogInput();
public byte getDigitalInput();
```

3. The *GripperPlayerDevice* class

3.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands - PS (p. 23)

```
public void setGripper(byte cmd, byte arg);
```

Methods – PS (p.23)

```
public byte getState();
public byte getBeams();
```

4. The *PositionPlayerDevice* class

4.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands - PS (p. 24 - 25)

```
public void setSpeed(int speed, int turnrate);
public void setSpeed(int speed, int turnrate,
                                int sideSpeed);
public void setMotorState(int state);
public void setSpeedMode(byte mode);
public void reset();
```

Methods – PS (p.24)

```
public int   getX()        { return x; }
public int   getY()        { return y; }
public short getHeading()  { return heading; }
public short getSpeed()    { return speed; }
public short getTurnrate() { return turnrate; }
public short getCompass()  { return compass; }
public byte  stall()       { return stalls; }
```

5. The *SonarPlayerDevice* class

5.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands - PS (p. 26)

```
public void setSonarPower(byte state);
```

Accessible variables – PS (p.26)

```
public int range[];
public int samplesCount;
```

6. The *LaserPlayerDevice* class

6.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands - PS (p. 27-28)

```
public void configureLaser(short startAngle,
                           short endAngle,
                           short resolution,
                           byte  intensity);
public void getConfiguration();
```

If the configuration of the laser changed, the following method would return true, otherwise – false;

```
public boolean isNewInfo ();
```

Methods – PS (p.27-28)

```
public int[] getRange();
public int[] getReflection();
public short getStartAngle();
public short getEndAngle();
public int   getResolution();
public int   getSamplesCount();
public short getIntensity();
```

7. The *VisionPlayerDevice* class

7.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Methods – PS (p.27-28)

```
public ColorChannel[] getColorChannels();
public ColorChannel   getColorChannel(int i);
```

ColorChannel class variables:

```
public short       index;
public short       noBlobs;
public ColorBlob[] blob;
```

ColorBlob class variables as in PS (p. 29)**:**

```
public int   color;
public int   area;
public short x;
public short y;
public short left;
public short right;
public short top;
public short bottom;
```

8. The *PtzPlayerDevice* class

8.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands – PS (p. 30)

```
public void setPTZ(short pan, short tilt, int zoom)
```

Methods – PS (p.30)

```
public short getPan();
public short getTilt();
public int   getZoom();
```

9. The *AudioPlayerDevice* class

9.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands – PS (p. 31)

```
public void produceSound(short freq, short amp,
                         short duration);
```

Methods – PS (p.31)

```
public int[] getFiveHighestFrequencies();
public int[] getFiveHighestAmplitudes();
```

10. The *BeaconPlayerDevice* class

10.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands – PS (p. 33)

```
public void setConfiguration(byte bitCount,
                             short bitWidth,
                             short zeroThresh,
                             short oneThresh);
public void getConfiguration();
```

If the configuration of the laser changed, the following method would return true, otherwise – false;

```
public boolean isNewInfo ();
```

Methods – PS (p.33)

```
public int      getBeaconCount();
public Beacon[] getBeacons();
public byte     getBitCount();
public short    getBitWidth();
public short    getZeroThresh();
public short    getOneThresh();
```

11. The *BroadcastPlayerDevice* class

11.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands – PS (p. 34-35)

```
public void sendMessage(String m);
public void receiveMessage();
```

Methods – PS (p.33)

```
public String  getMessage();
```

To diversify between cases when the new message is received and the old message still in the queue.

```
public void    setRead();
public boolean isRead();
```

12. The *SpeechPlayerDevice* class

12.1 Methods

Methods – PS (p.36)

```
public void say(String str);
```

13. The *GPSPlayerDevice* class

13.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Methods – PS (p.37)

```
public int getX();
public int getY();
public int getHeading();
```

14. The *VisionPlayerDevice* class

14.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands – PS (p. 38-39)

The variables of the *posInfo* array are the first 6 variables in table PS-5.26

```
public void setBeacon(int id, int[] posInfo);
```

Methods – PS (p.38)

```
public int getPX();
public int getPY();
public int getPA();
public int getUX();
public int getUY();
public int getUA();
```

```
public int getReserved();
```

15. The *TruthPlayerDevice* class

Device available only in Stage (only in simulations and not on the real hardware).

15.1 Methods

Method allowing to read the data manually, if *readAll()* of *PlayerClient* has not been called.

```
public void readData();
```

Commands

Teleports robot to a new location.

```
public void teleport(int x, int y);
public void teleport(int x, int y, int heading);
```

Methods – PS (p.38)

```
public boolean isTeleported();
public int     getX();
public int     getY();
public int     getHeading();
```