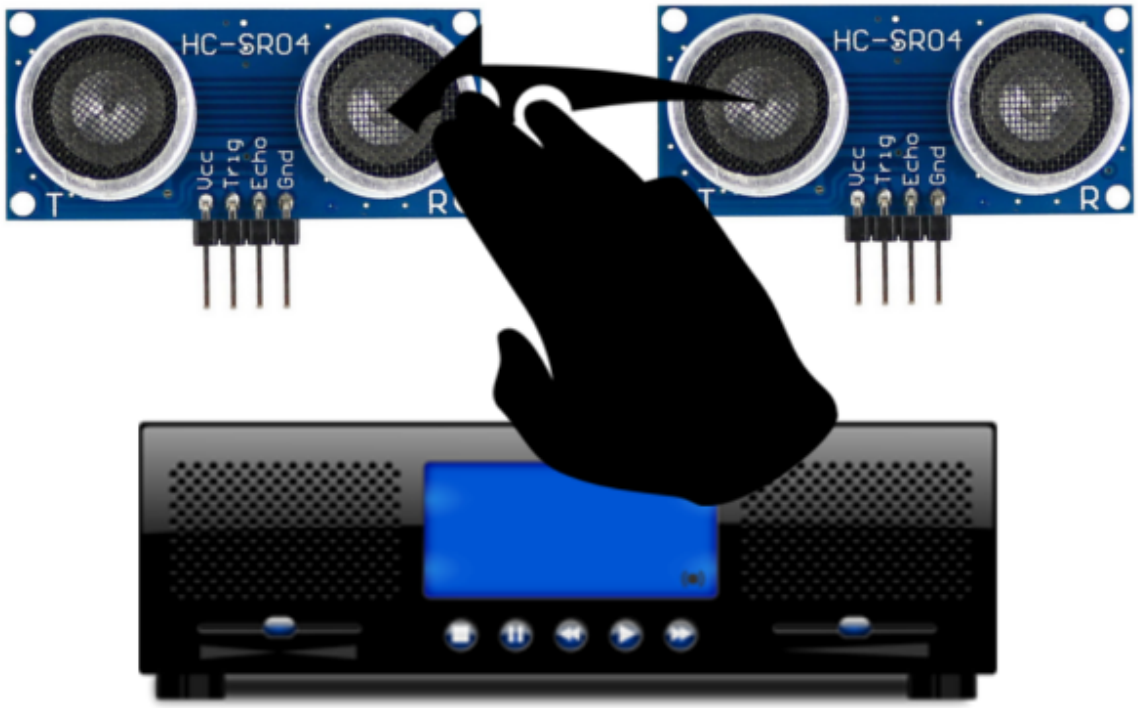
Athens week - UPM115

Arduino project

Simple gestures to control your audio reader



Introduction **3**
What material have we used ? 3
How does it work ? 3
Moves classification **4**
General gesture detection 4
Swipe gesture (previous and next song) 5
Up and down (volume control) 5
Pause detection 5
About gesture detection 6
Conception of a PC Audio interface **6**
Displayed window 6
Handling sound files 7
Communication between Arduino and PC **8**
Conclusion **9**
Our project 9
Greetings 9
Appendix **10**
Arduino source code 10
Audio PC application source code 15
Audio reader backend code 15
Interface source code 21

Introduction

Working hard during this Athens november session, we realized that it could be useful for lazy people like us to control our audio devices thanks to simple hand gestures. That is why we have decided to design a device which would be able to interact with a PC audio player to control it. We restricted ourselves to the most basic instructions that you would want to have, e.g. *playing*, *pausing*, moving *next* and *previous*, and finally set the volume *up* and *down*.

What material have we used ?

- 1 Arduino Nano chip
- 2 ultrasonic sensors HC-SR04
- 1 serial cable
- 1 PC to run the media player (developed with the Processing IDE)

How does it work ?

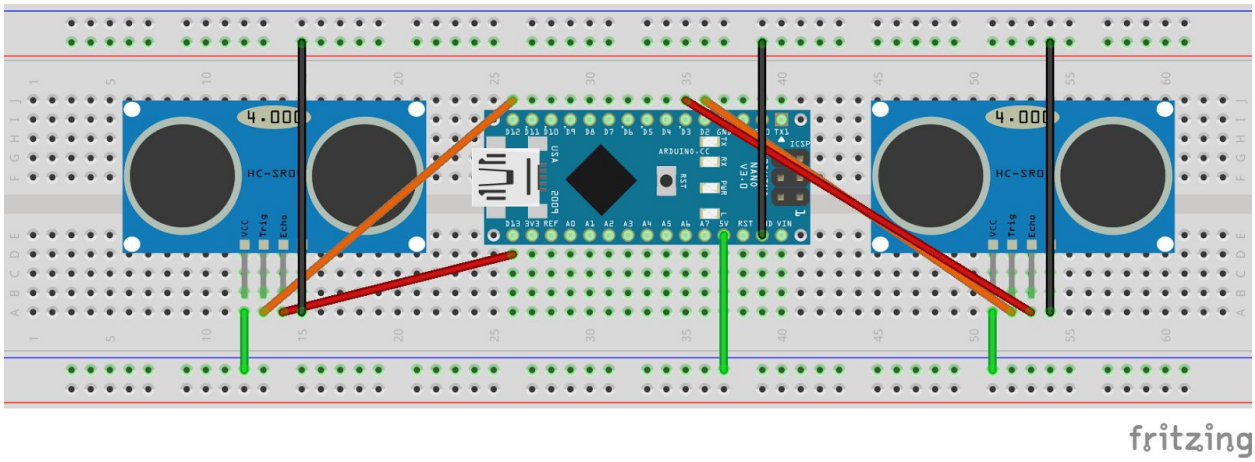


Figure 1 : schematics of our device

Basically, we use the ultrasonic sensors to measure the distance between them and a covering obstacle periodically. Using the information of two sensors, we have tried to classify different hand moves in front of our device. Once each particular move is classified on the chip, we can associate it to a command to make a particular action on the audio

player. The command is relayed from the *Arduino* to the PC application through a USB serial communication.

Moves classification

We implemented all in all 4 gestures to control the volume, pause or resume the song currently playing, and go to previous or next song.

General gesture detection

We used two HC-SR04 ultrasonic sensors to detect those gestures. They have two pins (apart from power supply), echo and trigger. Trigger is used to send an ultrasonic impulsion, and echo might measure its reflectance after a short delay, providing the distance information. Using two sensors allowed us to detect gestures in more than one dimension. The gestures we implemented for this project are as follows :

- ➔ Swipe gesture (previous and next song)
- ➔ Up and down (volume control)
- ➔ Holding hand on right sensor (pause)

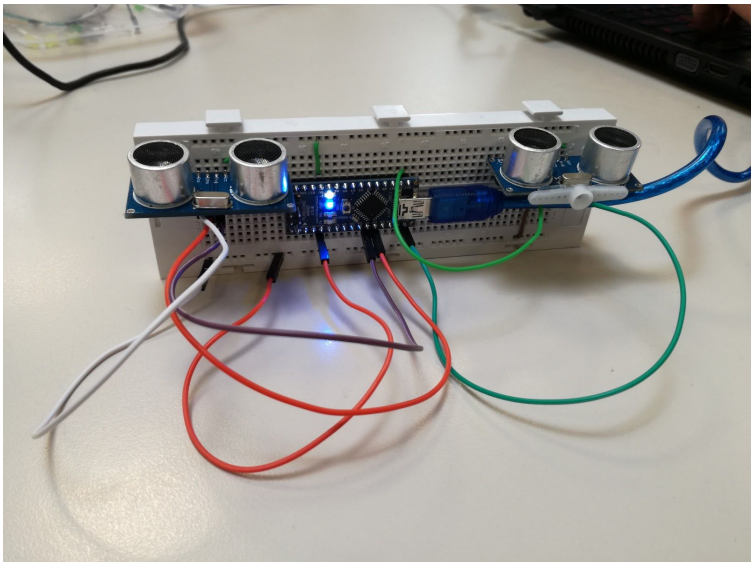


Figure 2 : The setup

Swipe gesture (previous and next song)

We used a swipe gesture to go to next or previous song. The user has to pass his hand overhead of the two sensors, from left to right to go to the next song, from right to left to go back to the previous song. In order to detect this gesture, we used the variables *lastLeftDetection* and *lastRightDetection*. They store the time when a last right/left obstacle was detected. If the difference between those two values is between **-4s** and **4s** the function signals a gesture detection. If it is <0, it is a *"previous"*, otherwise, it is a *"next"* command. After sending the 'p' or 'n' command to the interface, there is a **1s delay** before the next command.

We had to adjust this -4/4s value so that an user cannot send a command in holding his hands in front of both sensors.

Up and down (volume control)

To control the volume up and down, the user needs to hold his hand above the left sensor for **12 measure cycles**, counted by the counter *LeftCounter*. This 12 value was chosen so that you cannot go accidently change the volume while doing a swipe gesture. The cycles have an 25ms delay.

More than a simple gesture, volume control is a mode, represented by the boolean *volumeMode*, in order to avoid that other gestures are detected while controlling the volume.

Once in volume control mode, the user has to move his hand up and down in front of the left sensor. Every (-) **0.3cm**, (down) up counter is incremented. Every 3 measures, the program send the average direction to the interface, 'u' or 'd'.

Pause detection

The pause gesture is the simplest one. The user has to hold his hand in front of the right sensor for 16 measure cycles. Then, the program send 'g' to the interface.

About gesture detection

- ➔ Most of the values (delay, distance limit, …) used for gesture detection are experimental values. We tested our program a lot in order to find the optimal values for those parameters.
- ➔ The left sensor kept detecting sometimes a 4,91cm or 5,01cm far obstacle for no reason. We had to filter those values, after trying to switch the sensors, change the wires, etc…
- ➔ We had to find a lot of acceptable compromises between precision and usability. For example, we had to find the proper distance between the two sensors, so that they don't interfere and the user doesn't have to move his hand on an excessive distance.

Conception of a PC Audio interface

The PC application must have the following functions :

- ➔ Displaying an **interface** to show the audio player status
- ➔ Interacting with the PC operating system to use **sound files** and handling them
- ➔ Receiving audio requests through the USB **serial port**

We have used the *Processing* IDE as recommended, combined with a few libraries which allow us to design windows with basic widgets, manipulate sound files easily and using buffers to communicate with any PC serial port through which we need to transmit or receive data.

Displayed window

Figure 3 : the Audio player interface

The interface has been mainly designed to show the Audio reader state : we display the currently selected song into a text field, colour the two labels on the top (c.f. Figure 3) according to the playing/paused state (the left one is **green** when playing, while the other becomes **red** when the music is paused) and finally the volume in amplitude percentage with the cursor on the right.

We have added a few interface inputs, like the buttons that you can see (**<<**, **>>** and **Play/Pause**) or key event listeners (A press on **+** and **-** to adjust the volume) mainly for debugging as the goal of our project was to make these commands with gestures thanks to the designed device.

For all of this, the *G4P GUI editor* was really useful because it offers a WYSIWYG tool to design basic interfaces and generates the associated Java code in the application : all the widgets are drawn automatically as defined into the GUI file. Each widget interaction can be implemented thanks to callback methods.

Here are the G4P library widgets classes used in our interface.

- ➔ **GButton** to make all the buttons
- ➔ **GDropList** to display the currently playing song
- ➔ **GLabel** to draw the two “playing” / “paused” indicators at the top (‘>’ and ‘||’)
- ➔ **GSlider** for the volume cursor

Handling sound files

To load and handle sound files, we have used a Processing open source library called *Minim*. Each command you would want to do on a playing song can be easily done through instances of the *AudioPlayer* class. An instance of an *AudioPlayer* is created by the static method *Minim.loadFile(<your song file name>)*, and you just have to call the *play()* method on this instance to play the selected song, or call *pause()* to stop it temporarily.

Musics are stored into the project directory, in a sub-directory called **music** from which we list all the music files. To know the song position in the files list, we store the selected song index into an *integer* variable and increment/decrement it when necessary (moving to the next/previous song).

To show the playing/paused status, we have used the library method *isPlaying()* which returns **true** only if the selected song is actually playing in the *AudioReader*.

Finally, the volume can be adjusted only indirectly via the *setGain()* method. As the gain has an exponential link with the amplitude (or volume), we had to make conversions from a volume variation to the associated gain variation : for example, when we want to increment the volume with a value of 2% more, we simply have to call the *setGain()* method as follows :

$$newGain = 10 * log(current\ volume + delta(\%))$$

Communication between Arduino and PC

Once the command has been recognized by the *Arduino* thanks to the two ultrasonic sensors, it must be sent to the PC application through the serial port. To do this, we have set a serial communication, programmed in the *C++* code into the *Arduino* and *Java* into the application.

We have also defined a simple protocol which associates a character to each implemented command. You can find the details below.

- **g** : Play/pause
- **n** : Moves to the next song
- **p** : Resets the song at the beginning. If called two times in a 5s maximal interval, moves to the previous song
- **u** : Sets the volume up of a percentage value *
- **d** : Sets the volume down of the same absolute percentage value *

* : The percentage value can be adjusted thanks to a constant value defined at the beginning of the program (see the *PERCENT_BAR* attribute).

Once recognized by the *Arduino Nano*, the command is written through its serial port using *Serial.write()* and on the other side the application gets it using *Serial.readChar()*. We have implemented a main method called *process(char command)* which associates the received character to the *AudioReader* command to do.

Conclusion

Our project

To conclude the work of our end of Athens week, we think that our project was a good choice to have enough work in the given time while being not too much ambitious. It is a modular project which could be continued adding other gestures to handle the *Audio Player*. We had fun making gestures to move next/previous for example and the result is quite satisfying even if the gestures recognition could be done in a more natural way if we had time to implement a more complex one. The interface is also quite basic, as *G4P* is not a very evolved tool to make graphical windows but it has been sufficient for the experiments : we only had to show the *Audio Reader* state so *G4P* widgets fitted perfectly.

Our project could be improved by **four main guidelines** :

- Improving the already implemented features (gestures recognition, and the PC app interface) as mentioned above
- Adding new gestures, for example to set the elapsed time position into the currently playing song
- Adding outputs on our device, like a *LCD* screen which could display the currently playing song as what we have in cars radio systems
- Adding a remote connection between the device and the PC (*WPAN* or *WLAN*) like *Wi-Fi* or *Bluetooth*, to make a mobile controller

Greetings

First we thank the Athens coordinators to avoid us to focus on practical details during our week, and just enjoy our *Arduino* course while discovering the great city of Madrid.

We also thank the course teachers of course, for their availability, kindness and great teaching for our first time working on an *Arduino* chip. We really enjoyed the possibilities offered by this kind of devices.

To finish, thanks to all our European colleagues who followed courses in Madrid during this Athens week, and always were enthusiastic about sharing their culture and beliefs while offering us many activities in Madrid during their free time.

Appendix

Arduino source code

```
/* pins constants */
const byte TRIGGER_PIN_RIGHT = 12;
const byte ECHO_PIN_RIGHT = 13;
const byte TRIGGER_PIN_LEFT = 7;
const byte ECHO_PIN_LEFT = 8;

/* detection variables */
unsigned long lastLeftDetection = 0.0;
unsigned long lastRightDetection = 0.0;
unsigned long firstPauseDetection = 0.0;

/* constant needed for measure */
const unsigned long MEASURE_TIMEOUT = 25000UL; // 25ms = ~8m Ã  340m/s

/* counters */
int leftCounter = 0;
int noLeftCounter = 0;
int rightCounter = 0;
int upNb = 0;
int downNb = 0;
int measureCounter = 0;

/* current mode */
int volumeMode = 0;

/* measure needed for the volume control */
float initialLeftHeigth = 0;
```

```
/* speed of sound (mm/us) */
const float SOUND_SPEED = 340.0 / 1000;

void setup()
{
  /* initialize serial port */
  Serial.begin(9600);

  /* initialize pins */
  pinMode(TRIGGER_PIN_RIGHT, OUTPUT);
  pinMode(TRIGGER_PIN_LEFT, OUTPUT);
  digitalWrite(TRIGGER_PIN_RIGHT, LOW); // La broche TRIGGER doit Ãªtre Ã  LOW au repos
  digitalWrite(TRIGGER_PIN_LEFT, LOW);
  pinMode(ECHO_PIN_RIGHT, INPUT);
  pinMode(ECHO_PIN_LEFT, INPUT);
}

void loop() {

  //right impulsion
  digitalWrite(TRIGGER_PIN_RIGHT, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN_RIGHT, LOW);
  long measureRight = pulseIn(ECHO_PIN_RIGHT, HIGH, MEASURE_TIMEOUT);


  //left impulsion
  digitalWrite(TRIGGER_PIN_LEFT, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN_LEFT, LOW);
  long measureLeft = pulseIn(ECHO_PIN_LEFT, HIGH, MEASURE_TIMEOUT);

  float distanceLeftCm = measureLeft / 20.0 * SOUND_SPEED;
  float distanceRightCm = measureRight / 20.0 * SOUND_SPEED;

  detect(distanceLeftCm, 1, millis());
  detect(distanceRightCm, 0, millis());

  detectGestures();
```

```
  detectVolume(distanceLeftCm);
  detectPause();
  delay(25);
}


/* detect the user hand in 0-30cm, averages the downNb and upNb, and send u (up) or down
(down) to the server */
void detect(float distanceCm, int side, unsigned long tm)
{
  if(distanceCm<30.0 && distanceCm>1.0)
  {
    /* filter values that the left sensor keeps sending for no reason*/
    if(!(distanceCm>5.01 && distanceCm<5.02))
    {
      if(!(distanceCm>4.91 && distanceCm<4.92))
      {
        if(side==0)
        {
          lastRightDetection = tm;
          rightCounter++;
        }
        else
        {
          lastLeftDetection = tm;
          leftCounter++;
        }
      }
    }
  }
  else
  {
    noLeftCounter++;
    if(side==1 && noLeftCounter >= 15)
    {
      volumeMode = 0;
      initialLeftHeigth = 0;
      noLeftCounter = 0;
      leftCounter = 0;
```

```
    }
    if(measureCounter >= 3)
    {
      if(downNb > upNb)
      {
        for(int i = 0; i < downNb; i++)
        {
          Serial.write('d');
        }
      }
      else
      {
        for(int i = 0; i < upNb; i++)
        {
          Serial.write('u');
        }
      }
      downNb = 0;
      upNb = 0;
      measureCounter = 0;
    }
  }
}


/* detect the next/previous gesture and send n or p to the server */
void detectGestures()
{
  if(lastRightDetection > 0.1 && lastLeftDetection > 0.1)
  {
    signed long dTm = lastLeftDetection-lastRightDetection;
    if(dTm < 4000.0 && dTm > -4000.0)
    {
      if(!(dTm < 8.0 && dTm > -8.0) && volumeMode == 0)
      {
        if(dTm < 0)
        {
          Serial.write('n');
        }
```

```
        else
        {
          Serial.write('p');
        }
        lastRightDetection = 0.0;
        lastLeftDetection = 0.0;
        delay(1000);
      }
    }
  }
}

/* detect the volume gesture, increment upNb and downNb regarding the user gesture */
void detectVolume(float distanceLeftCm)
{
    if((leftCounter >= 12) && (volumeMode == 0))
  {
    leftCounter = 0;
    volumeMode = 1;
    initialLeftHeigth = distanceLeftCm;
  }

  if(volumeMode == 1 && distanceLeftCm > 0.1)
  {
    int dh = initialLeftHeigth-distanceLeftCm;
    if(dh > 0.3 && !(distanceLeftCm>5.01 && distanceLeftCm<5.02) && !(distanceLeftCm>4.91 &&
distanceLeftCm<4.92))
    {
      downNb++;
      initialLeftHeigth = distanceLeftCm;
      measureCounter++;
    }
    if(dh < -0.3 && !(distanceLeftCm>5.01 && distanceLeftCm<5.02) && !(distanceLeftCm>4.91 &&
distanceLeftCm<4.92))
    {
      upNb++;
      initialLeftHeigth = distanceLeftCm;
      measureCounter++;
    }
```

```
  }
}

/* detect pause gesture using the right sensor detection counter and send g to the server */
void detectPause()
{
  if(rightCounter >= 16)
  {
    Serial.write('g');
    rightCounter = 0;
    delay(1000);
  }
}
```

Audio PC application source code

Audio reader backend code

```
import processing.serial.* ;
import g4p_controls.* ;
import java.io.File ;
import ddf.minim.* ;

private final boolean SERIAL_CONNECTED = false ;
private final double ONE_PERCENT_VOL = 0.03981071706 ;
private final int PERCENT_VAR = 2 ;

// For serial communication with the controller
private Serial mp3Port ;

// Audio files reader variables
private String[] songsFileNames ;
private int selIndex = 0 ;
private String path ;
private int lastPreviousTime = 0 ;
```

```
private Minim minim ;
private AudioPlayer song ;


public void setup() {
  size(500, 150, JAVA2D) ;


  // Setting up the interface thanks to the GUI file
  createGUI() ;


  //Serial communication initialization
  printArray(Serial.list());
  String[] serialPorts = Serial.list() ;
  if (SERIAL_CONNECTED) mp3Port = new Serial(this, serialPorts[serialPorts.length - 1], 9600)
;


  // Sound files handling initialization
  minim = new Minim(this) ;
  path = sketchPath() + "/data/music/" ;
  File musicsDir = new File(path) ;
  songsFileNames = musicsDir.list() ;
  dropListSongs.setItems(songsFileNames, 0) ;
  song = minim.loadFile(path+songsFileNames[0]) ;
  volumeSlider.setEnabled(false) ; // Disabling mouse events on slider
  song.setGain(ampToGain(2.0f)) ;  // Setting volume at 50% (sound amplitude is between 0 and
4W approximately
  update() ;
}


public void draw() {
  if(SERIAL_CONNECTED) {
    if(mp3Port.available() > 0) {
```

```
      char command = mp3Port.readChar() ; // Command should be sent from the serial port as a
char
      process(command) ;
      println("Command received : "+command) ;
    }
  }
}


public void process(char command) {
  switch(command) {
    // Play / pause command
    case 'g' :
      playPause() ;
      break ;
    // Next song command
    case 'n' :
      next() ;
      break ;
    // Previous song command
    case 'p' :
      previous() ;
      break ;
    case 'u' :
      adjustVol(true) ;
      break ;
    case 'd' :
      adjustVol(false) ;
      break ;
  }
  update() ;
}
```

```
// Selects a song and prepares it to be played
public void selectSong(String name) {
  // To apply the last song state to the newly selected song
  boolean lastSongPlaying = song.isPlaying() ;
  float lastSongGain = song.getGain() ;
  // Selecting the new song and applying last song state to it
  minim.stop() ;
  minim = new Minim(this) ;
  song = minim.loadFile(path+name) ;
  song.setGain(lastSongGain) ;
  if(lastSongPlaying) song.play() ;
}

// Command n°1 : playing/pausing a song
private void playPause() {
  if(song.isPlaying())
    song.pause() ;
  else
    song.play() ;
}

// Command n°2 : going to the next song
private void next() {
  if(selIndex == songsFileNames.length - 1)
    selIndex = 0 ;
  else
    selIndex++ ;
  moveToSongIndex(selIndex) ;
}
```

```
// Command n°3 : going to the previous song
public void previous() {
  int currentTime = millis() ;
  // Goes to the previous song if clicked 2 times in less than 5s
  if(currentTime - lastPreviousTime < 5000) {
    if(selIndex == 0)
      selIndex = songsFileNames.length - 1 ;
    else
      selIndex-- ;
    moveToSongIndex(selIndex) ;
  }
  else // Otherwise it just goes to the current song beginning
    song.rewind() ;
  lastPreviousTime = currentTime ;
}

// Command n°4 : adjusting the volume
public void adjustVol(boolean up) {
  float currentGain = song.getGain() ;
  float currentVol = gainToAmp(currentGain) ;
  double delta ;
  if(up)
    delta = PERCENT_VAR * ONE_PERCENT_VOL ;
  else
    delta = - PERCENT_VAR * ONE_PERCENT_VOL ;
  float newVol = (float)(currentVol + delta) ;
  float newGain = ampToGain(newVol) ;
  if(newVol < ONE_PERCENT_VOL) {
    song.setGain(-80.0f) ;
```

```
    return ;
  }
  if(newVol > 99*ONE_PERCENT_VOL) {
    song.setGain(6.0f) ;
    return ;
  }
  song.setGain(newGain) ;
}

// Moves to a specified song index of the files list and plays it
private void moveToSongIndex(int index) {
  selectSong(songsFileNames[index]) ;
}

// Refreshes the interface of the Audio player
private void update() {
  // Droplist displayed element
  dropListSongs.setSelected(selIndex) ;
  // Playing/paused status displayed on the window
  if(song.isPlaying()) {
    labelPlaying.setLocalColorScheme(G4P.GREEN_SCHEME) ;
    labelPaused.setLocalColorScheme(G4P.BLUE_SCHEME) ;
  }
  else {
    labelPlaying.setLocalColorScheme(G4P.BLUE_SCHEME) ;
    labelPaused.setLocalColorScheme(G4P.RED_SCHEME) ;
  }
  // Volume displayed on the window
  float currentGain = song.getGain() ;
  float currentVolume = gainToAmp(currentGain) ;
```

```
  volumeSlider.setValue(currentVolume) ;
}

// Just computes volume power from a given gain value
private float gainToAmp(float gain) {
  return (float)Math.pow(10, gain/10.0) ;
}

private float ampToGain(float amp) {
  return (float)(10 * Math.log10(amp)) ;
}

public void keyPressed() {
  switch(key) {
    case '+' :
      process('u') ;
      break ;
    case '-' :
      process('d') ;
      break ;
  }
  update() ;
}
```

Interface source code

```
public void dropListSongs_click1(GDropList source, GEvent event) {
//_CODE_:dropListSongs:442952:

} //_CODE_:dropListSongs:442952:
```

```
public void buttonPlay_click(GButton source, GEvent event) { //_CODE_:buttonPlay:626981:
  process('g') ;
} //_CODE_:buttonPlay:626981:

public void buttonReset_click(GButton source, GEvent event) { //_CODE_:buttonReset:591978:
  process('p') ;
} //_CODE_:buttonReset:591978:

public void buttonNext_click(GButton source, GEvent event) { //_CODE_:buttonNext:412981:
  process('n') ;
} //_CODE_:buttonNext:412981:

public void volumeSlider_change(GSlider source, GEvent event) { //_CODE_:volumeSlider:738008:
} //_CODE_:volumeSlider:738008:

// Create all the GUI controls.
// autogenerated do not edit
public void createGUI(){
  G4P.messagesEnabled(false);
  G4P.setGlobalColorScheme(GCScheme.BLUE_SCHEME);
  G4P.setCursor(ARROW);
  surface.setTitle("MP3 player");
  dropListSongs = new GDropList(this, 45, 79, 313, 120, 5);
  dropListSongs.setItems(loadStrings("list_442952"), 0);
```

```
dropListSongs.addEventHandler(this, "dropListSongs_click1");
buttonPlay = new GButton(this, 160, 39, 80, 30);
buttonPlay.setText("PLAY / PAUSE");
buttonPlay.addEventHandler(this, "buttonPlay_click");
buttonReset = new GButton(this, 44, 39, 80, 30);
buttonReset.setText("<<");
buttonReset.addEventHandler(this, "buttonReset_click");
buttonNext = new GButton(this, 277, 40, 80, 30);
buttonNext.setText(">>");
buttonNext.addEventHandler(this, "buttonNext_click");
labelPlaying = new GLabel(this, 163, 10, 27, 25);
labelPlaying.setTextAlign(GAlign.CENTER, GAlign.MIDDLE);
labelPlaying.setText(">");
labelPlaying.setTextBold();
labelPlaying.setOpaque(false);
labelPaused = new GLabel(this, 210, 10, 30, 24);
labelPaused.setTextAlign(GAlign.CENTER, GAlign.MIDDLE);
labelPaused.setText("||");
labelPaused.setOpaque(false);
volumeSlider = new GSlider(this, 479, 18, 112, 100, 10.0);
volumeSlider.setRotation(PI/2, GControlMode.CORNER);
volumeSlider.setLimits(4.0, 4.0, 0.0);
volumeSlider.setNumberFormat(G4P.DECIMAL, 2);
volumeSlider.setOpaque(false);
```

```
  volumeSlider.addEventHandler(this, "volumeSlider_change");
  label0 = new GLabel(this, 430, 117, 33, 20);
  label0.setTextAlign(GAlign.CENTER, GAlign.MIDDLE);
  label0.setText("0%");
  label0.setOpaque(false);
  label1 = new GLabel(this, 430, 12, 47, 20);
  label1.setTextAlign(GAlign.CENTER, GAlign.MIDDLE);
  label1.setText("100%");
  label1.setOpaque(false);
}

// Variable declarations
// autogenerated do not edit
GDropList dropListSongs;
GButton buttonPlay;
GButton buttonReset;
GButton buttonNext;
GLabel labelPlaying;
GLabel labelPaused;
GSlider volumeSlider;
GLabel label0;
GLabel label1;
```