

# Server audio processing

## Overview

Since build [5.2.795](#) it is possible to extract raw audio data from published stream in PCM 16 bit format with a following processing on server side. For example, stream sound can be recorded to a file.

### Attention

A stream published should have at least one subscriber to decode and extract audio from it.

## Audio processing implementation

To intercept raw audio data, a Java class implementing `IDecodedPcmInterceptor` interface should be developed. The method `pcmDecoded()` of this class will receive decoded audio packets in PCM format, as byte array. Let's take a look to class implementation example to record raw audio from a stream into a WAV file:

## DecodedPcmInterceptorTest.java

```
// Package name should be strictly defined as com.flashphoner.sdk.media
package com.flashphoner.sdk.media;

// Import Flashphoner SDK packages as needed
import com.flashphoner.media.rtp.recorder.OutputFileType;
import com.flashphoner.media.utils.WaveUtil;
import com.flashphoner.sdk.setting.Settings;
import com.flashphoner.media.utils.FileNameUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
// Import standard Java packages as needed
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Timer;
import java.util.TimerTask;

/**
 * Custom decoded audio interceptor implementation example
 * The example records first 10 seconds of audio track from a stream
 * published to WAV file
 */
public class DecodedPcmInterceptorTest implements IDecodedPcmInterceptor{

    // Will log errors to server log.
    private static final Logger log =
    LoggerFactory.getLogger("DecodedPcmInterceptorTest");

    // File object to write data
    protected RandomAccessFile incomingRecorder;
    // Last audio packet timestamp
    private volatile long lastTs;
    // Sampling rate to write to a file header
    private int samplingRate;
    // Number of channels to write to a file header
    private int numChannels;
    // Timer to stop recording
    private Timer cancelTimer;
    // Stream name to form file name
    private String streamName;

    public DecodedPcmInterceptorTest() {
    }

    /**
     * Method to handle decoded audio
     * @param streamName - stream name
     * @param pcm - decoded audio data from packet as byte array
     * @param samplingRate - audio track sampling rate
     * @param numChannels - audio track number of channels
     * @param timestamp - audio packet timestamp
     */
    @Override
    public void pcmDecoded(String streamName, byte[] pcm, int
    samplingRate, int numChannels, long timestamp) {
        // Remember data to write to the file header
        updateSr(samplingRate);
        updateNumChannels(numChannels);
        // Remember the stream name and create the file to write
    }
}
```

```

        updateStreamName(streamName);
        // Start timeout to stop recording
        startCloseTimer();
        try {
            // Write audio data to the file
            recordIncoming(pcm, timestamp);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Method to start timer task which should stop recording after 10
    seconds
    */
    public void startCloseTimer() {
        if (cancelTimer == null) {
            cancelTimer = new Timer();
            cancelTimer.schedule(new TimerTask() {
                @Override
                public void run() {
                    try {
                        close();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            }, 10000);
        }
    }

    /**
     * Method to store audio sampling rate to write it to the file header
    */
    public void updateSr(int samplingRate) {
        if (this.samplingRate == 0) {
            this.samplingRate = samplingRate;
        }
    }

    /**
     * Method to store audio channels number to write it to the file header
    */
    public void updateNumChannels(int numChannels) {
        if (this.numChannels == 0) {
            this.numChannels = numChannels;
        }
    }

    /**
     * Method to store the stream name and create the file to write
    */
    public void updateStreamName(String streamName) {
        if (this.streamName == null || this.streamName.isEmpty()) {
            // Create the file name
            this.streamName =
            FileNameUtils.adaptRecordName(streamName+".wav");
            try {
                // Create the file and reserve header space
                incomingRecorder = new
                RandomAccessFile(Settings.RECORD.getValue()+"/"+this.streamName, "rw");
            }
        }
    }

```

```

        incomingRecorder.write(new
byte[OutputFileType.WAV_HEADER_OFFSET]);
    } catch (IOException e) {
        log.error("Can't create DecodedPcmInterceptorTest, " +
e.getMessage());
    }
    log.info("Create DecodedPcmInterceptorTest");
}
}

/**
 * Method to write audio data to the file
 */
public void recordIncoming(byte[] data, long ts) throws IOException {
    incomingRecorder.write(data);
    lastTs = ts;
}

/**
 * Method to close file
 */
private void close() throws IOException {
    // Write header to the file
    writeHeader();
    try {
        // Close the file
        incomingRecorder.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    log.info("Close DecodedPcmInterceptorTest");
}

/**
 * Method to write header to the beginning of the file
 */
protected void writeHeader() throws IOException {
    // Get the file size
    int size = (int) incomingRecorder.length();
    // Form the header
    byte[] header = WaveUtil.getPcmWaveHeader((size -
OutputFileType.WAV_HEADER_OFFSET), samplingRate, numChannels);
    // Write the header to the beginning of the file
    incomingRecorder.seek(0);
    incomingRecorder.write(header);
}
}
}

```

Then the class should be compiled into byte code. To do this, create folder tree according to class package name

```
mkdir -p com/flashphoner/sdk/media
```

and execute the command

```
javac -cp /usr/local/FlashphonerWebCallServer/lib/wcs-
core.jar:/usr/local/FlashphonerWebCallServer/lib/slf4j-api-1.6.4.jar
```

```
./com/flashphoner/sdk/media/DecodedPcmInterceptorTest.java
```

Now, pack the code compiled to jar file

```
jar -cf testPcmInterceptor.jar ./com/flashphoner/sdk/media/*.class
```

and copy this file to WCS libraries folder

```
cp testPcmInterceptor.jar /usr/local/FlashphonerWebCallServer/lib
```

To use custom frames interceptor class, set its package name and folder to record WAV files to the following parameters in [flashphoner.properties](#) file

```
decoded_pcm_interceptor=com.flashphoner.sdk.media.DecodedPcmInterceptorTest  
record=/usr/local/FlashphonerWebCallServer/records
```

and restart WCS.

## A separate folder for custom Java libraries

Since build [5.2.1512](#), custom layout Java libraries (jar files) should be placed to the folder

```
/usr/local/FlashphonerWebCallServer/lib/custom
```

```
cp testlayout.jar /usr/local/FlashphonerWebCallServer/lib/custom
```

This folder is kept while updating WCS to a newer builds. A jar files do not need to be copied again after updating.

## Testing

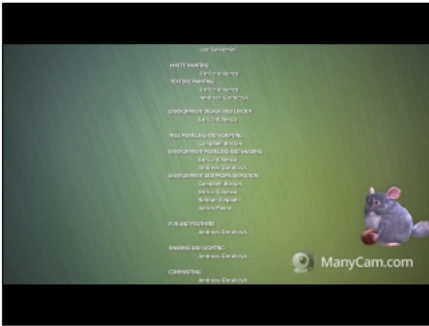
### 1. Publish a test stream in Two Way Streaming

example [https://test1.flashphoner.com:8444/client2/examples/demo/streaming/two\\_way\\_streaming/two\\_way\\_streaming.html](https://test1.flashphoner.com:8444/client2/examples/demo/streaming/two_way_streaming/two_way_streaming.html), where [test1.flashphoner.com](#) is WCS

server address

## Two-way Streaming

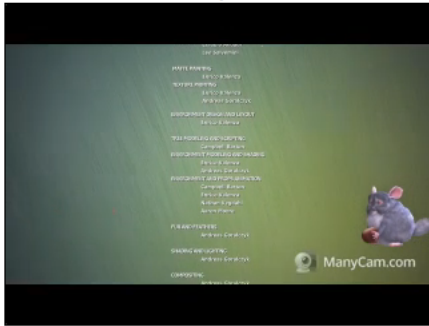
Local



test      Stop

PUBLISHING

Player



test      Stop      Available

PLAYING

wss://test1.flashphoner.com:8443      Disconnect

ESTABLISHED

2. Check if WAV file existst in `/usr/local/FlashphonerWebCallServer/records/` folder

```
total 1812
-rw-r--r-- 1 root root 1854764 Oct 22 16:01 test-1badb540-1445-11eb-9dfd-614c13aa0eb7.wav
```