This document describes all new class files.

ContentServer.java

```
package netserv.apps.activestreaming.server;

import java.io.*;
import java.net.*;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;
import java.util.StringTokenizer;
import java.util.logging.Logger;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.*;

import org.mortbay.jetty.*;
import org.mortbay.jetty.servlet.*;

/**
 * Content Server : All the request comes to this Servlet. This Servlet
 * redirects all the request to NetServ nodes, if any. otherwise creates a
 * direct connection to streaming server.
 */
public class ContentServer extends HttpServlet {

        private static final long serialVersionUID = 1L;
        // Map for maintaining current streaming files.
        private Map<String, VideoStreamer> streamingMap = new HashMap<String, VideoStreamer>();
        private static String SERVER_IP;
        private static String STREAM_SERVER_IP;
        private static String NETSERV_NODE_IP;
        private static final String CONTENT_SERVER_PORT = "8088";
        private static final String NETSERV_NODE_PORT = "8888";
        private static final String STREAM_SERVER_PORT = "8080";
        public static final String LOCALROOT = "./sample";

        private static final Logger log = Logger.getLogger(ContentServer.class
                        .getName());

        // NetServ Configuration
        static final String SERVER_PROPERTIES = "./server.properties";

        // These static variables are also being set in the server.properties file
        public static String WEBROOT = "./";
        public static String NSIS_TRIGGER = "./netserv-trigger";
        public static String GEOIP_FILE = "./GeoLiteCity.dat";
        public static int MODULE_LIFETIME = 24000;
        public static double THRESHOLD_DISTANCE = 1000.0; // kilometers ?

        ContentSingleton singleton = ContentSingleton.getInstance();

        /**
         * Servlet context initializer
         */
        public void init(ServletConfig config) throws ServletException {
                super.init(config);
                singleton.readTranslationToHashMap();
        }

        public void doGet(HttpServletRequest request, HttpServletResponse response) {
                String file = request.getParameter("file");
                String mode = request.getParameter("mode");
                final String client = request.getRemoteAddr();

                log.info("Request for " + file + " from " + request.getRemoteAddr());
                if (file == null) {
                        try {
                                response.getWriter().write("File not found !");
                                return;
                        } catch (IOException e2) {
                                e2.printStackTrace();
                        }
                }

                // NetServ using NSIS
                // sendNetServSignal(client, file, mode, response);
                // NetServ using simple setup
                simpleSetup(client, file, mode, response);

        }

        private void simpleSetup(final String client, String file, String mode,
                        HttpServletResponse response) {
                if (mode != null && mode.equalsIgnoreCase("live")) {
                        log.info("simpleSetup : Sending to NetServ Node .. mode=Live");
                        sendVLCVideo(file, response, true, false, NETSERV_NODE_IP);
                } else {
                        sendVLCVideo(file, response, false, false, NETSERV_NODE_IP);
        }
    }

    private void sendNetServSignal(final String client, String file,
            String mode, HttpServletResponse response) {
        String netServNode = getNetServNode(client);
        if (netServNode == null) {
            // First request
            Thread signalThread = new Thread(new Runnable() {
                public void run() {
                    String node = null;
                    // 5 times: sleep 2 second and probe the install
                    for (int i = 0; i < 5; i++) {
                        sendNetServSetup(client);
                        try {
                            Thread.sleep(2000);
                        } catch (InterruptedException e) {
                            log.severe("signalThread : Error while sleeping: "
                                    + e.toString());
                        }
                        node = sendNetServProbe(client);
                        if (node != null) {
                            break;
                        }
                    }
                    try {
                        Thread.sleep(MODULE_LIFETIME * 1000 - 5000);
                    } catch (InterruptedException e) {
                        log.severe("signalThread : Error while sleeping: "
                                + e.toString());
                    }

                    if (node != null)
                        sendNetServTeardown(node);
                }
            });
            signalThread.start();

            if (mode != null && mode.equalsIgnoreCase("live")) {
                log.info("sendNetServSignal : Sending directly to streaming server .. mode=Live");
                sendVLCVideo(file, response, true, true, null);
            } else {
                log.info("sendNetServSignal : Sending directly to streaming server..");
                sendVLCVideo(file, response, false, true, null);
            }
        } else {
            // NetServ node is present
            if (mode != null && mode.equalsIgnoreCase("live")) {
                log.info("sendNetServSignal : Sending to NetServ Node .. mode=Live");
                sendVLCVideo(file, response, true, false, netServNode);
            } else {
                log.info("sendNetServSignal : Sending to NetServ Node..");
                sendVLCVideo(file, response, false, false, netServNode);
            }
        }
    }

    /**
     * Returns the nearest NetServ Node for a client.
     *
     * @param client
     * @return
     */
    private String getNetServNode(String client) {
        // Get all NetServ nodes
        Set<String> nodes = singleton.getNodes();

        String netServNode = null;
        double globalDistance = THRESHOLD_DISTANCE;
        try {
            Iterator<String> it = nodes.iterator();
            log.info("getNetServNode : Total nodes found - " + nodes.size());
            while (it.hasNext()) {
                String node = (String) it.next();
                double distance = singleton.calc_distance(netServNode, client);
                if (distance < globalDistance) {
                    globalDistance = distance;
                    netServNode = node;
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            log.severe("getNetServNode : Error calculating NetServ Node distance.");
        }

        if (globalDistance < THRESHOLD_DISTANCE) {
            log.info("getNetServNode : Found NetServ node " + netServNode
                    + " [" + client + ", " + globalDistance + "]");
            return netServNode + ":" + NETSERV_NODE_PORT;
        } else {
            return null;
        }
    }
    /**
     * Send NetServ setup signal
     *
     * @param client
     *             - IP address of the client node
     * @return Boolean value determining whether operation was successful or not
     */
    private boolean sendNetServSetup(String client) {
        String s;
        String command = NSIS_TRIGGER
                + " "
                + client
                + " -s -user jae -id NetServ.apps.ActiveStreaming_1.0.0 -url http://netserv-
server/modules/activestreaming.jar -ttl "
                + MODULE_LIFETIME;
        log.info("sendNetServSetup : " + command);
        Process p;
        try {
            p = Runtime.getRuntime().exec(command);
            BufferedReader stdInput = new BufferedReader(new InputStreamReader(
                    p.getInputStream()));

            while ((s = stdInput.readLine()) != null) {
                // output will look like: 2 0 0
                if (s.equals("2 0 0"))
                    return true;
                else
                    return false;
            }
        } catch (IOException e) {
            log.severe("sendNetServSetup : Error running trigger setup \n"
                    + e.toString());
        }
        return false;
    }

    /**
     * Send NetServ probe.
     */
    private String sendNetServProbe(String client) {
        String command = NSIS_TRIGGER
                + " "
                + client
                + " -p -user jae -id NetServ.apps.ActiveStreaming_1.0.0 -probe 2";
        log.info("sendNetServProbe : " + command);
        BufferedReader stdInput;
        Process p;
        try {
            p = Runtime.getRuntime().exec(command);
            stdInput = new BufferedReader(new InputStreamReader(
                    p.getInputStream()));
            // output will look like:
            // 1.2.3.4 ACTIVE (for working nodes)
            // 1.2.3.4 NOT PRESENT (for non-working nodes)
            String s;
            while ((s = stdInput.readLine()) != null) {
                StringTokenizer st = new StringTokenizer(s);
                String ipAddr = null;
                String status = null;
                ipAddr = st.nextToken();
                if (ipAddr != null) {
                    status = st.nextToken();
                }
                if (status.equals("ACTIVE")) {
                    if (singleton.addNode(ipAddr))
                        log.info("sendNetServProbe : Adding NetServ node "
                                + ipAddr);
                    return ipAddr;
                }
            }
        } catch (IOException e) {
            log.severe("sendNetServProbe : NSIS trigger not present");
            log.severe("sendNetServProbe : Error adding NetServ node"
                    + e.toString());
        }
        return null;
    }

    /**
     * Removes the NetServ Node mapping for given IP address.
     *
     * @param ipAddr
     * @return
     */

    private boolean sendNetServTeardown(String ipAddr) {
        String command = NSIS_TRIGGER + " " + ipAddr
                + " -r -user jae -id NetServ.apps.ActiveStreaming_1.0.0";
        log.info("sendNetServTeardown : " + command);
        try {
            Runtime.getRuntime().exec(command);
            singleton.removeNode(ipAddr);
            log.info("sendNetServTeardown : Removing NetServ node " + ipAddr);
            return true;
```

```
		} catch (IOException e) {
			log.info("sendNetServTeardown : Error removing node list: "
					+ e.toString());
		}
		return false;
	}

	/**
	 * Returns a VLC plugin embedded HTML page.
	 * 
	 * @param file
	 *            - streaming file name
	 * @param response
	 *            - http response object
	 * @param isLive
	 *            - whether live mode or not
	 * @param isDirect
	 *            - whether there direct connection to streaming server
	 */
	private void sendVLCVideo(String file, HttpServletResponse response,
			boolean isLive, boolean isDirect, String server) {
		String url = null;
		if (isDirect) {
			url = this.directURL(file);
		} else {
			if (isLive) {
				url = this.netServNodeURL(file, server);
				url += "&mode=live";
			} else {
				url = this.netServNodeURL(file, server);
				url += "&mode=vod";
			}
		}
		try {
			PrintWriter pr = response.getWriter();
			pr.write("<html>");
			pr.write("<head><title>NetServ Active Streaming</title></head>");
			pr.write("<body>");
			pr.write("<div id=\"content\" align=\"center\">");
			pr.write("<h2>NetServ Active Streaming</h2>");

			pr.write("<div align=\"center\"><a href=\"http://"
					+ ContentServer.SERVER_IP + ":"
					+ ContentServer.CONTENT_SERVER_PORT + "/stream/?file="
					+ file + "&mode=live\">View Live</a> </div>");

			pr.write("<embed type=\"application/x-vlc-plugin\" name="
					+ file
					+ " autoplay=\"yes\" loop=\"no\" width=\"680\" height=\"460\" target=\""
					+ url + "\"" + " />");
			pr.write("</div>");
			pr.write("<br />");
			pr.write("</body>");
			pr.write("</html>");
			pr.flush();
		} catch (IOException e1) {
			e1.printStackTrace();
		}
	}

	/**
	 * Used to send HTML5 video instead of VLC plugin. Tested it on Chrome 10
	 * 
	 * @param file
	 * @param response
	 */
	private void sendHTML5Video(String file, HttpServletResponse response,
			String server) {
		try {
			PrintWriter pr = response.getWriter();
			pr.write("<html>");
			pr.write("<head><title>NetServ Active Streaming</title></head>");
			pr.write("<body>");
			pr.write("<div id=\"content\" align=\"center\">");
			pr.write("<h2>NetServ Active Streaming</h2>");
			pr.write("<div id=\"netserv-video\">");
			pr.write("<video id=\"demo-video\" controls>");
			pr.write("<source src=\"" + this.netServNodeURL(file, server)
					+ "\"" + "type=\"video/ogg\" />");
			pr.write("</video>");
			pr.write("</div></div>");
			pr.write("<br />");
			pr.write("</body>");
			pr.write("</html>");
			pr.flush();
		} catch (IOException e1) {
			e1.printStackTrace();
		}
	}

	/**
	 * 
	 * @param file
	 * @return
	 */
	private String netServNodeURL(String file, String server) {
		String newurl = "";
		newurl = "http://" + server + ":" + NETSERV_NODE_PORT
				+ "/stream-cdn/?url=" + directURL(file);
		return newurl;
	}

	private String directURL(String file) {
		String newurl = "";
		newurl = "http://" + STREAM_SERVER_IP + ":" + STREAM_SERVER_PORT;
		return newurl;
	}

	/**
	 * Returns the first non looping ipv4 address.
	 * 
	 * @return
	 */
	private static InetAddress getFirstNonLoopbackAddress() {
		Enumeration<NetworkInterface> en = null;
		try {
			en = NetworkInterface.getNetworkInterfaces();
		} catch (SocketException e) {
			log.severe("Error getting network interface.." + e.toString());
		}
		while (en.hasMoreElements()) {
			NetworkInterface i = (NetworkInterface) en.nextElement();
			for (Enumeration en2 = i.getInetAddresses(); en2.hasMoreElements();) {
				InetAddress addr = (InetAddress) en2.nextElement();
				if (!addr.isLoopbackAddress()) {
					if (addr instanceof Inet4Address) {
						return addr;
					}
				}
			}
		}
		return null;
	}

	/**
	 * Only on Linux systems, embedded VLC streaming server is supported media
	 * files are streamed on - http://ipaddress:8080/stream/file_1. mp4 - "asf"
	 * mux container 2. ogv - "ogg" mux container
	 * 
	 * @param args
	 * @throws Exception
	 */
	public static void main(String[] args) throws Exception {
		// starting streaming service
		if (args.length == 2) {
			String file = args[0];
			SERVER_IP = args[1];
			// netserv node ip is calculated dynamically by NSIS
			NETSERV_NODE_IP = SERVER_IP;
			STREAM_SERVER_IP = SERVER_IP;
			log.info("Content Server address : " + SERVER_IP);
			// VideoStreamer.playMedia(LOCALROOT + "/" + file, file,
			// "asf",SERVER_IP, STREAM_SERVER_PORT);
			// starting jetty server
			Server server = new Server(
					Integer.parseInt(ContentServer.CONTENT_SERVER_PORT));
			Context root = new Context(server, "/", Context.SESSIONS);
			root.addServlet(new ServletHolder(new ContentServer()), "/stream/*");
			log.info("Content Server started..");
			server.start();
			server.join();
		} else {
			log.severe("Usage: ContentServer <movie_file_name> <content server ip> <netserv node ip>");
		}

	}
}
```

VideoStreamer.java

```java
package netserv.apps.activestreaming.server;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.HashSet;
import java.util.Set;

import uk.co.caprica.vlcj.player.MediaPlayerFactory;
import uk.co.caprica.vlcj.player.headless.HeadlessMediaPlayer;

/**
 * Embedded VLC Player, uses libvlc for creating a headless player or a
 * streaming server.
 *
 */
class VideoStreamer {
        private static HeadlessMediaPlayer mediaPlayer = null;
        private static String serverAddress;
        private static String serverPort;
        public static Set<String> Files = new HashSet<String>();
        public static final VideoStreamer PLAYER;

        static {
                PLAYER = new VideoStreamer();
        }

        /**
         * A private constructor which creates a single instance of the headless
         * player.
         */
        private VideoStreamer() {
                MediaPlayerFactory mediaPlayerFactory = new MediaPlayerFactory();
                mediaPlayer = mediaPlayerFactory.newMediaPlayer();
        }

        public static void playMedia(String media, String file, String mux,
                        String ip, String port) {
                Files.add(file);
                VideoStreamer.serverAddress = ip;
                VideoStreamer.serverPort = port;
                file = formatHttpStream(file, mux);
                mediaPlayer.playMedia(media, file);
                System.out.println("Starting streaming.. " + media + " with options "
                                + file);

                // when the streaming gets over we need to remove the file
        }

        public void closePlayer() {
                mediaPlayer.release();
        }

        private static String formatHttpStream(String file, String mux) {
                StringBuilder sb = new StringBuilder(60);
                sb.append(":sout=#standard{access=http,mux=" + mux + ",");
                sb.append("dst=");
                sb.append(serverAddress);
                sb.append(':');
                sb.append(serverPort + "/stream/" + file);
                sb.append("}");
                return sb.toString();
        }
}
```

ActiveStreamNode.java

```
package netserv.apps.activestreaming.module;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.FileInputStream;
import java.io.InputStream;
import java.io.OutputStream;

import java.net.URL;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.util.logging.Logger;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.mortbay.jetty.Server;
import org.mortbay.jetty.servlet.Context;
import org.mortbay.jetty.servlet.ServletHolder;

/**
 * This is the Active Streaming Node main class. We need to save the stream from
 * Streaming server and pass it to connected client.
 */

public class ActiveStreamNode extends HttpServlet {

	private static final long serialVersionUID = 1L;
	ActiveStreamMap singleton = ActiveStreamMap.getInstance();
	/**
	 * if(inMemoryBuffer): make one connection to stream server else: make 'T'
	 * connections to stream server
	 */
	public static boolean inMemoryBuffer = false;
	// total circular buffer frames
	public static final int FRAMES = 10000;
	// frame size
	public static final int FRAME_SIZE = 1024;
	private static final Logger log = Logger.getLogger(ActiveStreamNode.class
			.getName());

	public void doPost(HttpServletRequest request, HttpServletResponse response) {
		this.doGet(request, response);
	}

	public void doGet(HttpServletRequest request, HttpServletResponse response) {
		final String url = request.getParameter("url");
		final String mode = request.getParameter("mode");

		if (url == null && mode == null) {
			try {
				response.getWriter().print(
						"URL and mode are both required parameters.. ");
				response.getWriter().close();
			} catch (IOException e) {
				e.printStackTrace();
			}
			return;
		}

		CacheVideo cacheVideo = singleton.addURL(url);
		long total = cacheVideo.getTotalByteSaved();
		File cacheFile = cacheVideo.getCacheFile();

		synchronized (cacheVideo.activeConn) {
			if (!mode.equalsIgnoreCase("live"))
				cacheVideo.activeConn += 1;
		}

		if (cacheVideo.getState() == CacheVideo.INITIAL) {
			log.info("Request received for streaming " + url + " from "
					+ request.getRemoteAddr());
			VideoWriter writer = new VideoWriter(cacheVideo);
			Thread writerThread = new Thread(writer);
			writerThread.setPriority(Thread.MAX_PRIORITY);
			writerThread.start();
			cacheVideo.writerInstance = writer;
			cacheVideo.setState(CacheVideo.LIVE);
			// this.addClient(response, cacheVideo);
			this.serveURL(cacheVideo, response);

		} else if (cacheVideo.activeConn < CacheVideo.STORAGE_THRESHOLD) {
			log.info("GET : Active connections less than threshold,read from video buffer");
			// this.addClient(response, cacheVideo);
			this.serveURL(cacheVideo, response);
		} else if (cacheVideo.activeConn == CacheVideo.STORAGE_THRESHOLD) {
			log.info("GET : Threshold reached, trigger local storage.."
					+ cacheFile.getName());
			cacheVideo.setStoreCache(true);
```

```
            this.serveURL(cacheVideo, response);
        } else if (mode.equalsIgnoreCase("live")) {
            log.info("GET : Going to live mode.." + cacheFile.getName());
            serveFromFile(cacheVideo, response, total);
        } else if (cacheVideo.activeConn > CacheVideo.STORAGE_THRESHOLD) {
            log.info("GET : Serving via local cache from starting.."
                    + cacheFile.getName());
            serveFromFile(cacheVideo, response, 0);
        } else if (mode.equalsIgnoreCase("stop")) {
            log.info("GET : Stop writer");
            cacheVideo.writerInstance.stop_write();
        }
    }

    /**
     * Used to add clients to the circular buffer
     *
     * @param response
     * @param cv
     */
    public void addClient(HttpServletResponse response, CacheVideo cv) {
        File localFile = cv.getCacheFile();
        response.setHeader("Content-Disposition", "inline; filename="
                + localFile.getName());
        response.setHeader("Cache-Control", "no-cache");
        response.setHeader("Expires", "-1");

        Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
        int readerPos = 0;
        long readerFrame = 0;
        synchronized (cv.writerFrame) {
            readerPos = getReaderPos(ActiveStreamNode.FRAMES / 2, cv);
            readerFrame = getReaderFrame(ActiveStreamNode.FRAMES / 2, cv);
        }

        log.info("Reader: Buffer Position: " + readerPos + " & Frame : "
                + readerFrame);
        while (cv.getState() != CacheVideo.LOCAL) {
            if (readerFrame < cv.writerFrame) {
                try {
                    OutputStream out_stream = response.getOutputStream();
                    out_stream.write(cv.videoBuffer, readerPos * FRAME_SIZE,
                            FRAME_SIZE);
                    log.info("Writing to stream");
                } catch (org.mortbay.jetty.EofException e) {
                    log.warning("Reader : Browser connection closed !");
                    break;
                } catch (IOException e) {
                    log.severe("Reader: Error while flusing data to client :(");
                    e.printStackTrace();
                    break;
                }
                readerFrame++;
            } else {
                try {
                    log.info("Reader: Going to sleep :(");
                    Thread.sleep(3000);
                    synchronized (cv.writerFrame) {
                        readerPos = getReaderPos(ActiveStreamNode.FRAMES / 2,
                                cv);
                        readerFrame = getReaderFrame(
                                ActiveStreamNode.FRAMES / 2, cv);
                    }
                } catch (InterruptedException e) {
                    log.info("Reader: Thread interrupted from sleep");
                    e.printStackTrace();
                    break;
                }
            }
        }
    }

    /**
     * Circular Buffer Helper method, to get real array position for reader.
     *
     * @param windowSize
     * @param cv
     * @return
     */

    private int getReaderPos(long windowSize, CacheVideo cv) {
        long pos = 0;
        int r1 = FRAMES / 2, r2 = FRAMES;
        long frame = cv.writerFrame % ActiveStreamNode.FRAMES;
        if (!cv.onceFilled) {
            if (0 <= frame && frame < r1) {
                pos = 0;
            } else if (r1 <= frame && frame < r2) {
                pos = frame - windowSize;
            }
        } else {
            if (frame < windowSize) {
                pos = ActiveStreamNode.FRAMES - (windowSize - frame);
            } else {
```

```
                        pos = frame - windowSize;
                }
        }
        return (int) pos;
}

/**
 * Circular Buffer Helper method, to calculate the reader start Frame number
 *
 * @param windowSize
 * @param cv
 * @return
 */
private long getReaderFrame(long windowSize, CacheVideo cv) {
        long pos = 0;
        int r1 = FRAMES / 2, r2 = FRAMES;
        if (!cv.onceFilled) {
                if (0 <= cv.writerFrame && cv.writerFrame < r1) {
                        pos = 0;
                } else if (r1 <= cv.writerFrame && cv.writerFrame < r2) {
                        pos = cv.writerFrame - windowSize;
                }
        } else {
                pos = cv.writerFrame - windowSize;
        }
        return pos;
}

/**
 * Used to serve the video request directly from the origin server.
 *
 * @param cv
 * @param response
 */

public void serveURL(CacheVideo cv, HttpServletResponse response) {
        InputStream in = null;
        try {
                File localFile = cv.getCacheFile();
                response.setHeader("Content-Disposition", "inline; filename="
                                + localFile.getName());
                response.setHeader("Cache-Control", "no-cache");
                response.setHeader("Expires", "-1");

                log.info("serveURL : Serving directly from origin stream");
                byte[] buf = new byte[FRAME_SIZE];
                int count = 0;
                URL urlstream = cv.getVideoURL();
                in = urlstream.openStream();
                OutputStream out_stream = response.getOutputStream();
                while ((count = in.read(buf)) > 0) {
                        out_stream.write(buf, 0, count);
                }
        } catch (org.mortbay.jetty.EofException e) {
                log.warning("serveURL : Browser connection closed !");
        } catch (IOException e) {
                log.warning("serveURL : IOExcetion");
                e.printStackTrace();
        }

        try {
                if (in != null)
                        in.close();
        } catch (IOException e) {
                log.warning("serveURL : Error closing IO streams");
                e.printStackTrace();
        }
}

/**
 * Used to serve request from local repository
 */
public void serveFromFile(CacheVideo cv, HttpServletResponse response,
                long skipBytes) {
        log.info(cv.toString());
        File localFile = cv.getCacheFile();
        response.setHeader("Content-Disposition", "inline; filename="
                        + localFile.getName());
        response.setHeader("Cache-Control", "no-cache");
        response.setHeader("Expires", "-1");
        try {
                FileInputStream in = new FileInputStream(localFile);
                FileChannel in_channel = in.getChannel();
                OutputStream out = response.getOutputStream();

                // Copy the contents of the file to the output stream
                byte[] buf = new byte[FRAME_SIZE];
                ByteBuffer buf_wrap = ByteBuffer.wrap(buf);

                long count = 0;
                if (skipBytes > 0) {
                        log.info("ServeFromFile : Moving file position to " + skipBytes);
                        in_channel.position(skipBytes - 5 * FRAME_SIZE);
                        while (in_channel.position() >= cv.getTotalByteSaved())
```

```
                                        Thread.yield();

                                while (cv.getState() != CacheVideo.LOCAL) {
                                        count = in_channel.read(buf_wrap);
                                        if (count > 0) {
                                                out.write(buf);
                                        }
                                        buf_wrap.clear();
                                }
                        } else {
                                while ((count = in_channel.read(buf_wrap)) >= 0) {
                                        out.write(buf, 0, (int) count);
                                        buf_wrap.clear();
                                }
                        }
                        log.info("ServeFromFile : Reached here, STATE " + cv.getState());
                        in_channel.close();
                        in.close();
                        out.flush();
                        out.close();
                } catch (org.mortbay.jetty.EofException e) {
                        log.warning("ServeFromFile : Browser connection closed !");
                } catch (IOException e) {
                        log.warning("ServeFromFile : While writing to browser's stream.");
                }
        }

        /**
         * Writer Thread main class
         *
         */
        protected class VideoWriter implements Runnable {
                boolean stopped;
                CacheVideo cv;

                private VideoWriter(CacheVideo cv) {
                        this.cv = cv;
                }

                public void run() {
                        log.info("Writer: Contacting streaming server & saving locally.. ");
                        byte[] buf = new byte[FRAME_SIZE];
                        int count = 0;
                        FileOutputStream out_file = null;
                        File cacheFile = cv.getCacheFile();
                        try {
                                out_file = new FileOutputStream(cacheFile);
                                URL urlstream = cv.getVideoURL();
                                cv.originInputStream = urlstream.openStream();
                                int frame = 0;
                                while ((count = cv.originInputStream.read(buf)) > 0) {
                                        if (inMemoryBuffer) {
                                                if (frame < FRAMES) {
                                                        System.arraycopy(buf, 0, cv.videoBuffer, frame
                                                                        * FRAME_SIZE, buf.length);
                                                } else {
                                                        frame = 0;
                                                        System.arraycopy(buf, 0, cv.videoBuffer, frame
                                                                        * FRAME_SIZE, buf.length);
                                                        if (!cv.onceFilled) {
                                                                cv.onceFilled = true;
                                                                log.info("Writer: Video buffer is once filled !");
                                                        }
                                                }

                                                synchronized (cv.writerFrame) {
                                                        cv.writerFrame++;
                                                        frame++;
                                                }
                                        }
                                        if (cv.isStoreCache()) {
                                                out_file.write(buf, 0, count);
                                                out_file.flush();
                                                cv.incrementTotalBytes(count);
                                        }

                                        if (stopped) {
                                                break;
                                        }

                                }
                        } catch (IOException e) {
                                log.severe("Writer: Problem occurred in writing to file :(");
                                e.printStackTrace();
                        } finally {
                                // check if we want to save the file
                                if (cv.isStoreCache()) {
                                        cv.setState(CacheVideo.LOCAL);
                                } else {
                                        // remove the file from cache
                                        cacheFile.delete();
                                }
                        }
                }
```

```
		synchronized void stop_write() {
			stopped = true;
			notify();
		}
	}

	/**
	 * Main method, NetServ Node runs on Port: 8888 and /stream-cdn/ servlet
	 * context.
	 *
	 * @param args
	 */
	public static void main(String[] args) {
		Server server = new Server(8888);
		Context root = new Context(server, "/", Context.SESSIONS);
		root.addServlet(new ServletHolder(new ActiveStreamNode()),
				"/stream-cdn/*");
		log.info("NetServ node started..");
		try {
			server.start();
			server.join();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

CacheVideo.java

```java
package netserv.apps.activestreaming.module;

import java.io.File;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;

import netserv.apps.activestreaming.module.ActiveStreamNode.VideoWriter;


/**
 * Each video URL instantiates this class object, This class stores all the
 * information related to a URL.
 */

public class CacheVideo {
        /* cache video file states */
        public static final int NOT_PRESENT = 0;
        public static final int INITIAL = 1;
        public static final int LIVE = 2;
        public static final int LOCAL = 3;
        public static final int STORAGE_THRESHOLD = 1;
        public static final String LOCALROOT = "./cached-video";

        public InputStream originInputStream;
        public VideoWriter writerInstance;

        // video buffer
        public byte[] videoBuffer;
        public Long writerFrame = new Long(0);
        public boolean onceFilled;

        // total active connections for this video url
        public Integer activeConn = new Integer(0);

        private URL videoURL = null;
        private int state;
        private File cacheFile;
        private boolean storeCache;
        private long totalByteSaved = 0;

        public CacheVideo(String url) {
                videoBuffer = new byte[ActiveStreamNode.FRAMES
                                * ActiveStreamNode.FRAME_SIZE];
                try {
                        videoURL = new URL(url);
                } catch (MalformedURLException e) {
                        e.printStackTrace();
                }
                cacheFile = initializeURL(url);
                this.state = INITIAL;
        }

        /**
         * Ideally we should check whether the URL is streaming. We assuming the
         * stream is present.
         */
        private File initializeURL(String url) {

                String host = videoURL.getHost();
                String path = videoURL.getPath();
                path = path.replaceAll("~", "");
                String cache = LOCALROOT + "/" + host + path;
                cacheFile = new File(cache);
                cacheFile.getParentFile().mkdirs();
                return cacheFile;
        }

        public boolean isStoreCache() {
                return storeCache;
        }

        public void setStoreCache(boolean storeCache) {
                this.storeCache = storeCache;
        }

        public URL getVideoURL() {
                return videoURL;
	}

	public void setVideoURL(URL videoURL) {
		this.videoURL = videoURL;
	}

	public int getState() {
		return state;
	}

	public void setState(int currentState) {
		this.state = currentState;
	}

	public File getCacheFile() {
		return cacheFile;
	}

	public void setCacheFile(File cacheFile) {
		this.cacheFile = cacheFile;
	}

	public long getTotalByteSaved() {
		return totalByteSaved;
	}

	public void setTotalByteSaved(long totalByteSaved) {
		this.totalByteSaved = totalByteSaved;
	}

	public void incrementTotalBytes(long count) {
		this.totalByteSaved += count;
	}
}
```