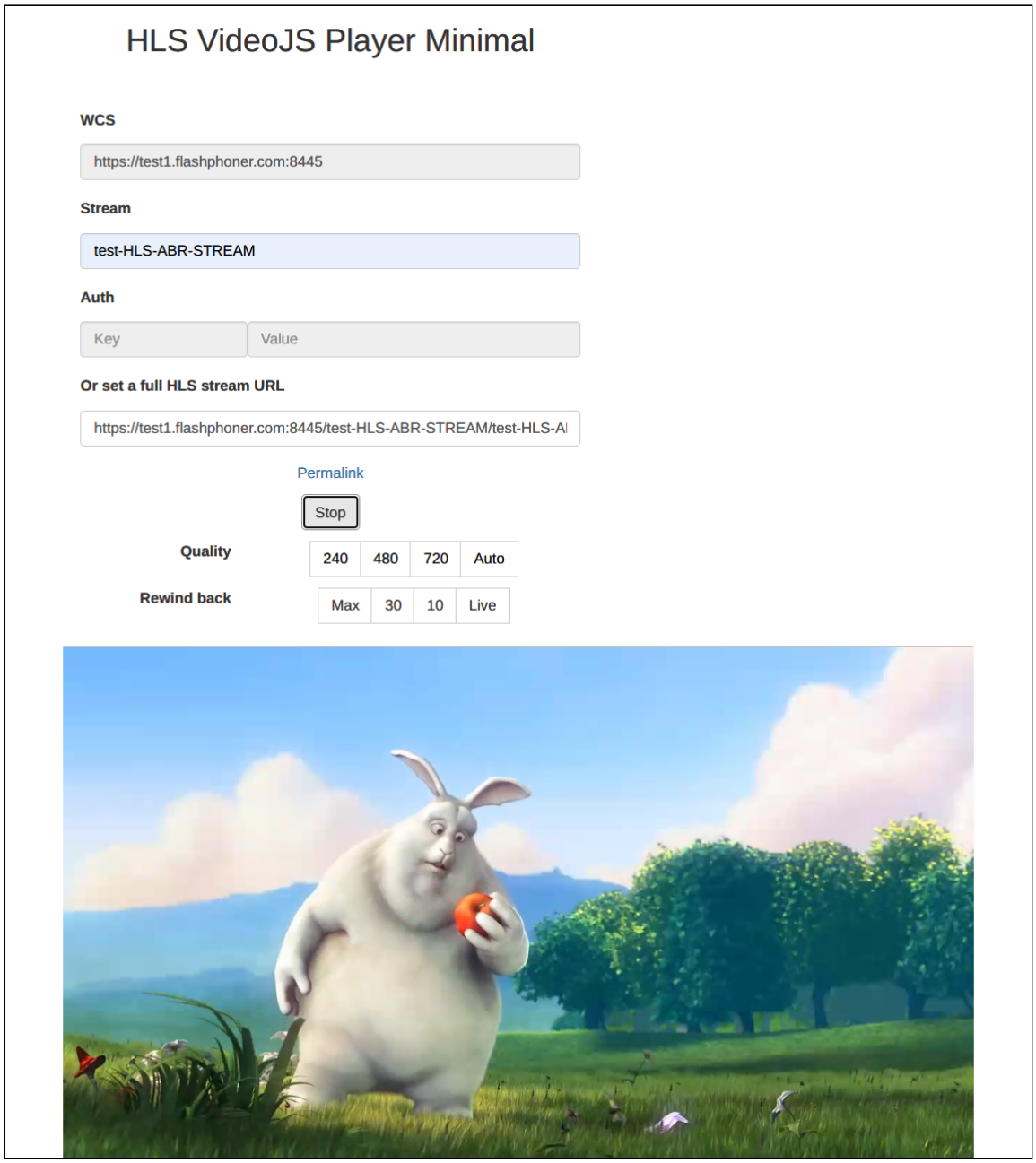
HLS VideoJS Player

Overview

The example shows how to convert stream published on WCS server to HLS and play it in browser using VideoJS library. HLS stream cut starts automatically when stream is requested by HLS URL, for example `https://test1.flashphoner.com:8445/test/test.m3u8` on the screenshot below.

The example allows to choose VideoJS version to play. VideoJS 8 supports Low Latency HLS if server allows to play this format



Since build 2.0.244, the example supports the following parameters:

- `version` - VideoJS version to use: `videojs7` or `videojs8`
- `src` - stream full HLS URL to play, should be encoded with URI, for example `https%3A%2F%2Ftest1.flashphoner.com%3A8445%2Ftest%2Ftest.m3u8`
- `autoplay` - automatically play the HLS URL, in this case all the input fields and buttons are hidden: `false` (by default) or `true`

The player URL example with parameters as displayed on the screenshot above (the link is available in `Permalink` field)

```
https://test1.flashphoner.com:8444/client2/examples/demo/streaming/hls-player/hls-player.html?
```

```
version=videojs7&src=https%3A%2F%2Ftest1.flashphoner.com%3A8445%2Ftest-HLS-ABR-STREAM%2Ftest-HLS-ABR-STREAM.m3u8
```

The player URL example with autoplay enabled

```
https://test1.flashphoner.com:8444/client2/examples/demo/streaming/hls-player/hls-player.html?version=videojs7&src=https%3A%2F%2Ftest1.flashphoner.com%3A8445%2Ftest-HLS-ABR-STREAM%2Ftest-HLS-ABR-STREAM.m3u8&autoplay=true
```

In this case the stream should be played automatically with audio muted. A viewer should use a loud slider in the player interface to unmute audio.

The code of the example

The source code is available on server by the following path:

```
/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/hls-player
```

- `hls-player.css` - player page styles file
- `hls-player.html` - player page
- `hls-player.js` - player launch script
- `player-page.html` - common player page elements for three HLS playback examples

The `videojs7` and `videojs8` subfolders contain two VideoJS versions respectively:

- `video.js` - the player library script (http://videojs.com/, Apache License Version 2.0)
- `video.min.js` - the player library script (minimized)
- `videojs-contrib-quality-levels.js` - quality levels plugin (VideoJS 7 only)
- `video-js.css` - the player library styles file

The example may be tested using the fiollowing URL:

```
https://host:8444/client2/examples/demo/streaming/hls-player/hls-player.html
```

Where `host` is WCS server address

Analyzing the code

To analyze the code get `hls-player.js` file version with hash `b19f637`, which is available here and may be downloaded in build 2.0.248.

1. Loading the player page

code

```
const loadPlayerPage = function() {
    if (videojsVersion) {
        hideItem("videojsInputForm");
        loadVideoJS(videojsVersion);
    } else {
        if (autoplay) {
            console.warn("No VideoJS version set, autoplay disabled");
            autoplay = false;
        }
        let videojsInput = document.getElementById("videojsInput");
        for (videojsType in VIDEOJS_VERSION_TYPE) {
            let option = document.createElement("option");
            let videojsFolder = "";
            switch (videojsType) {
                case 'VIDEOJS7':
                    videojsFolder = VIDEOJS_VERSION_TYPE.VIDEOJS7;
                    break;
                case 'VIDEOJS8':
                    videojsFolder = VIDEOJS_VERSION_TYPE.VIDEOJS8;
                    break;
            }
            option.text = videojsFolder;
            option.value = videojsFolder;
            videojsInput.appendChild(option);
        }

        setHandler("videojsBtn", "click", onVideojsBtnClick);
    }
}
```

2. Loading the VideoJS chosen version

code

```
const loadVideoJS = function(version) {
    if (version) {
        videojsVersion = version;
        let playerPage = document.getElementById("playerPage");
        loadFile(version + "/video.js", "text/javascript").then( data  => {
            console.log("HLS library loaded successfully", data);
            loadStyles(version, playerPage);
        }).catch( err => {
            setText("videojsError", "Can't load VideoJS library");
            console.error(err);
        });
    }
}
```

3. Loading the chosen VideoJS styles

code

```
const loadStyles = function(version, playerPage) {
    if (version) {
        loadFile(version + "/video-js.css", "stylesheet").then ( data => {
            console.log("HLS library stylesheet loaded successfully", data);
            if (version === VIDEOJS_VERSION_TYPE.VIDEOJS7) {
                loadQualityPlugin(version, playerPage);
            } else {
                hideItem("videojsInputForm");
                loadPage("player-page.html", "playerPage", initPage);
            }
        }).catch( err => {
            playerPage.innerHTML = "Can't load VideoJS library stylesheet";
            playerPage.setAttribute("class", "text-danger");
            console.error(err);
        });
    }
}
```

4. Loading quality levels plugin for VideoJS 7

code

```
const loadQualityPlugin = function(version, playerPage) {
    if (version) {
        loadFile(version + "/videojs-contrib-quality-levels.js",
"text/javascript").then( data => {
            console.log("HLS quality levels plugin loaded successfully",
data);
            hideItem("videojsInputForm");
            loadPage("player-page.html", "playerPage", initPage);
        }).catch( err => {
            setText("videojsError", "Can't load VideoJS quality levels
plugin");
            console.error(err);
        });
    }
}
```

5. The player HTML page initializing

code

```
const initPage = function() {
    if (playSrc) {
        setValue("fullLink", decodeURIComponent(playSrc));
    } else if (autoplay) {
        console.warn("No HLS URL set, autoplay disabled");
        autoplay = false;
    }
    let remoteVideo = document.getElementById('remoteVideo');
```

```
    if (autoplay) {
        // There should not be any visible item on the page unless player
        hideAllToAutoplay();
        // The player should use all available page width
        setUpPlayerItem(true);
        // The player should be muted to automatically start playback
        player = initVideoJsPlayer(remoteVideo, true);
        playBtnClick();
    } else {
        // No autoplay, all the forms and buttons should be visible
        setText("header", "HLS VideoJS Player Minimal");
        displayCommonItems();
        setUpButtons();
        enablePlaybackStats();
        // The player should have a maximum fixed size
        setUpPlayerItem(false);
        // The player can be unmuted because user should click Play button
        player = initVideoJsPlayer(remoteVideo, false);
    }
}
```

6. Player initializing

`videojs()` code

The following parameters are passed to the player:

- `video` - div tag to play the stream on the page
- `playsinline: true` - play a video on the page without automatic full screen switching (ignored in iOS Safari)
- `playbackRates` - playback rates list
- `liveui: true` - enables rewind (DVR) interface
- `liveTracker` - live playback thresholds setup
- `fill: true` - scale the player to div tag dimensions
- `muted` - enable or disable (for autoplay) audio
- `html5.vhs.limitRenditionByPlayerDimensions: false` - do not limit playback quality by div tag size

```
const initVideoJsPlayer = function(video, muted) {
    let videoJsPlayer = null;
    if (video) {
        video.className = "video-js vjs-default-skin";
        videoJsPlayer = videojs(video, {
            playsinline: true,
            playbackRates: [0.1, 0.25, 0.5, 1, 1.5, 2],
            liveui: true,
            liveTracker: {
                trackingThreshold: LIVE_THRESHOLD,
                liveTolerance: LIVE_TOLERANCE
```

```
            },
            fill: true,
            muted: muted,
            html5: {
                vhs: {
                    limitRenditionByPlayerDimensions: false
                }
            }
        });
        console.log("Using VideoJs " + videojs.VERSION);
        if (Browser.isSafariWebRTC() && Browser.isiOS()) {
            // iOS hack when using standard controls to leave fullscreen mode
            let videoTag = getActualVideoTag();
            if(videoTag) {
                setWebkitFullscreenHandlers(videoTag, false);
            }
        }
    }
    return videoJsPlayer;
}
```

7. HLS stream URL forming

`encodeURIComponent()` code

If authentication key and token are set, they will be included to stream URL

```
const getVideoSrc = function(src) {
    let videoSrc = src;
    if (validateForm()) {
        let streamName = getValue('playStream');
        streamName = encodeURIComponent(streamName);
        videoSrc = getValue("urlServer") + '/' + streamName + '/' +
streamName + '.m3u8';
        let key = getValue('key');
        let token = getValue("token");
        if (key.length > 0 && token.length > 0) {
            videoSrc += "?" + key + "=" + token;
        }
    }
    setValue("fullLink", videoSrc);
    return videoSrc;
}
```

8. Player launching

`player.on()`, `player.play()` code

```
const playBtnClick = function() {
    let videoSrc = getVideoSrc(getValue("fullLink"));
    if (videoSrc) {
        player.on('loadedmetadata', function() {
            console.log("Play with VideoJs");
```

```
            player.play();
        });
        ...
        player.src({
            src: videoSrc,
            type: "application/vnd.apple.mpegurl"
        });
        onStarted();
    }
}
```

9. `playing` event handler

`player.on()` code

```
const playBtnClick = function() {
     let videoSrc = getVideoSrc(getValue("fullLink"));
     if (videoSrc) {
         ...
         player.on('playing', function() {
            console.log("playing event fired");
            displayPermalink(videoSrc);
            if (player.liveTracker) {
                if (!player.liveTracker.isLive()) {
                    // A cratch to display live UI for the first subscriber
                    liveUIDisplay();
                }
                if (player.liveTracker.atLiveEdge()) {
                    // Unlock backward buttons when seeked to live edge
                    toggleBackButtons(true);
                    // Stop live UI startup timer
                    stopLiveUITimer();
                }
            }
            initQualityLevels(player);
            displayQualitySwitch();
        });
        ...
    }
}
```

10. Enable rewind (DVR) interface for the first subscriber

`player.liveTracker.seekToLiveEdge()` code

```
const liveUIDisplay = function() {
    stopLiveUITimer()
    if (player && player.liveTracker) {
        liveUITimer = setInterval(function() {
            if (!player.liveTracker.isLive() &&
player.liveTracker.liveWindow() > LIVE_THRESHOLD) {
                // Live UI is not displayed yet, seek to live edge to display
                player.liveTracker.seekToLiveEdge();
```

```
            }
        }, LIVE_UI_INTERVAL)
    }
}
```

11. Get available qualities list

`player.qualityLevels()` code

```
const initQualityLevels = function(player) {
    if (player && !qualityLevels.length) {
        let playerQualityLevels = player.qualityLevels();
        if (playerQualityLevels) {
            let qualityDiv = document.getElementById("qualityBtns");
            let qualityLevel;
            for (let i = 0; i < playerQualityLevels.length; i++) {
                qualityLevel = QualityLevel(playerQualityLevels,
playerQualityLevels[i].height, i, qualityDiv);
                qualityLevels.push(qualityLevel);
            }
            if (qualityLevels.length) {
                qualityLevel = QualityLevel(playerQualityLevels,
QUALITY_AUTO, -1, qualityDiv);
                qualityLevels.push(qualityLevel);
            }
        }
    }
}
```

12. Rewind button click action

`player.seekable()`, `player.currentTime()` code

```
const backBtnClick = function(event) {
    if (player != null && player.liveTracker) {
        toggleBackButtons(false);
        let seekable = player.seekable();
        let backTime = -1;
        if (event.target.id.indexOf("10") !== -1) {
            backTime = player.currentTime() - 10;
        } else if (event.target.id.indexOf("30") !== -1) {
            backTime = player.currentTime() - 30;
        }
        if (backTime < 0) {
            backTime = seekable ? seekable.start(0) : player.currentTime();
        }
        player.currentTime(backTime);
    }
}
```

13. Live button click action

`player.liveTracker.seekToLiveEdge()` code

```
const liveBtnClick = function() {
    if (player != null && player.liveTracker) {
        player.liveTracker.seekToLiveEdge();
        toggleBackButtons(true);
    }
}
```

14. Quaity button click action

`player.qualityLevels().selectedIndex_`, `player.qualityLevels().trigger()` code

```
const qualityBtnClick = function(button, playerQualityLevels, index) {
    if (playerQualityLevels && playerQualityLevels.length) {
        let currentIndex = playerQualityLevels.selectedIndex_;
        for (let i = 0; i < playerQualityLevels.length; i++) {
            let qualityLevel = playerQualityLevels[i];
            if (index === -1 || i === index) {
                qualityLevel.enabled = true;
            } else if (i === index) {
                qualityLevel.enabled = true;
                currentIndex = index;
            } else {
                qualityLevel.enabled = false;
            }
        }
        playerQualityLevels.selectedIndex_ = currentIndex;
        playerQualityLevels.trigger({ type: 'change', selectedIndex:
currentIndex });
    }
    button.style.color = QUALITY_COLORS.SELECTED;
    qualityLevels.forEach(item => {
        if (item.button.id !== button.id) {
            item.button.style.color = QUALITY_COLORS.AVAILABLE
        }
    });
}
```

15. Playback stopping

`player.dispose()` code

This method removes the div container tag where player was initialized from the page

```
const stopBtnClick = function() {
    if (player != null) {
        console.log("Stop VideoJS player");
        stopLiveUITimer();
        player.dispose();
    }
```

```
    onStopped();
}
```

16. Dispose qualities list when player is stopped

code

```
const disposeQualityLevels = function() {
    qualityLevels.forEach(level => {
        if (level.button) {
            level.button.remove();
        }
    });
    qualityLevels = [];
}
```

17. New `div` container tag creation after previous player was removed

code

```
const createRemoteVideo = function(parent) {
    let remoteVideo = document.createElement("video");
    remoteVideo.id = "remoteVideo";
    remoteVideo.controls="controls";
    remoteVideo.autoplay="autoplay";
    remoteVideo.type="application/vnd.apple.mpegurl";
    remoteVideo.className = "video-js vjs-default-skin";
    remoteVideo.setAttribute("playsinline","");
    remoteVideo.setAttribute("webkit-playsinline","");
    parent.appendChild(remoteVideo);
    player = initVideoJsPlayer(remoteVideo, autoplay);
}
```

18. Getting an available playback statistics from HTML5 video tag

`HTML5Stats` code

```
const PlaybackStats = function(interval) {
    const playbackStats = {
        interval: interval || STATS_INTERVAL,
        timer: null,
        stats: null,
        start: function() {
            let video = getActualVideoTag();

            playbackStats.stop();
            stats = HTML5Stats(video);
            playbackStats.timer = setInterval(playbackStats.displayStats,
playbackStats.interval);
            setText("videoWidth", "N/A");
            setText("videoHeight", "N/A");
```

```
            setText("videoRate", "N/A");
            setText("videoFps", "N/A");
            showItem("stats");
        },
        stop: function() {
            if (playbackStats.timer) {
                clearInterval(playbackStats.timer);
                playbackStats.timer = null;
            }
            playbackStats.stats = null;
            hideItem("stats");
        },
        displayStats: function() {
            if (stats.collect()) {
                let width = stats.getWidth();
                let height = stats.getHeight();
                let bitrate = stats.getBitrate();
                let fps = stats.getFps();

                setText("videoWidth", width);
                setText("videoHeight", height);

                if (bitrate !== undefined) {
                    setText("videoRate", Math.round(bitrate));
                }
                if (fps !== undefined) {
                    setText("videoFps", fps.toFixed(1));
                }
            }
        }
    };
    return playbackStats;
}
```