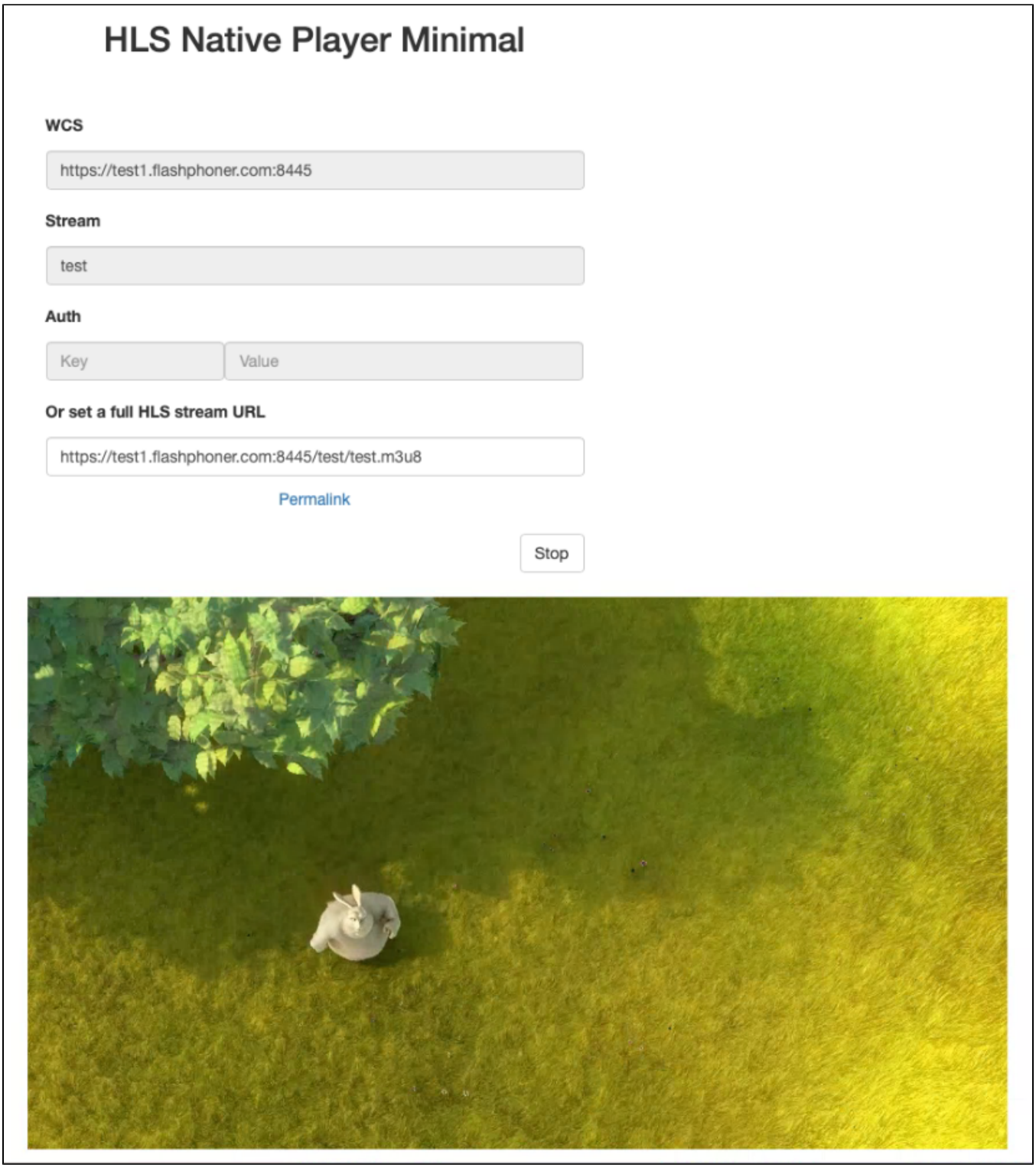
HLS Native Player

Overview

The example shows how to convert stream published on WCS server to HLS and play it in browser supporting HLS playback natively, on video tag level. 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



Since build2.0.244, the example supports the following parameters:

- 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-native/hls-native.html?src=https%3A%2F%2Ftest1.flashphoner.com%3A8445%2Ftest%2Ftest.m3u8
```

The player URL example with autoplay enabled

```
https://test1.flashphoner.com:8444/client2/examples/demo/streaming/hls-native/hls-native.html?src=https%3A%2F%
2Ftest1.flashphoner.com%3A8445%2Ftest%2Ftest.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 can be accessed on server by the following path:

/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/hls-native

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

The example can be tested using the fiollowing URL:

https://host:8888/client2/examples/demo/streaming/hls-js-player/hls-js-player.html

Where host is WCS server address

Analyzing the code

To analyze the code get hls-native.js file version with hash 1703e13 which is available hereand can be downloaded in build 2.0.244.

1. Loading the player page

code

```
const loadPlayerPage = function() {
    loadPage("../hls-player/player-page.html", "playerPage", initPage );
}
```

2. The player HTML page initializing

code

If browser does not support HLS native playback, player will not be initialized and a warning will be displayed

```
const initPage = function() {
    if (playSrc) {
        setValue("fullLink", decodeURIComponent(playSrc));
    } else if (autoplay) {
        console.warn("No HLS URL set, autoplay disabled");
        autoplay = false;
    }
    remoteVideo = document.getElementById('remoteVideo');
    if (remoteVideo.canPlayType('application/vnd.apple.mpegurl') && Browser.isSafariWebRTC()) {
        console.log("Using Native HLS player");
        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
            initVideoPlayer(remoteVideo, true);
            playBtnClick();
        } else {
            setText("header", "HLS Native 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
            initVideoPlayer(remoteVideo, false);
        }
    } else {
        setText("notifyFlash", "Your browser doesn't support native HLS playback");
        disableItem("applyBtn");
        toggleInputs(false);
    }
}
```

3. Video tag initializing to play

code

```
const initVideoPlayer = function(video, muted) {
    if (video) {
        video.style.backgroundColor = "black";
        video.muted = muted;
        if (Browser.isiOS()) {
            // iOS hack when using standard controls to leave fullscreen mode
            setWebkitFullscreenHandlers(video);
        }
    }
}
```

4. HLS stream URL forming

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;
}
```

5. Player launching

code

```
const playBtnClick = function() {
    let videoSrc = getVideoSrc(getValue("fullLink"));
    if (videoSrc) {
        remoteVideo.onloadedmetadata = () => {
            console.log("Play native HLS");
            remoteVideo.play();
            onStarted();
        };
        remoteVideo.onplaying = () => {
            console.log("playing event fired");
            displayPermalink(videoSrc);
        };
        remoteVideo.src = videoSrc;
    }
}
```

6. Playback stopping

code

```
const stopBtnClick = function() {
    if (remoteVideo != null) {
        console.log("Stop HTML5 player");
        remoteVideo.pause();
        remoteVideo.currentTime = 0;
        remoteVideo.removeAttribute('src');
        remoteVideo.load();
    }
    onStopped();
}
```

7. Getting an available playback statistics from HTML5 video tag

HTML5Statscode

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

            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;
}
```