

Embed Player

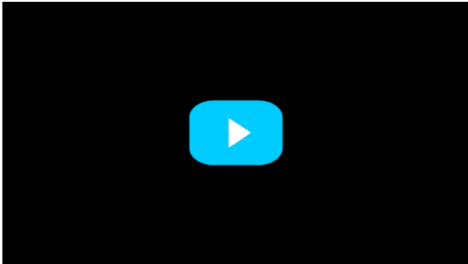
Example of a player that is embedded on a web page

This example may be used to embed player to the web page for live streams from web and IP cameras playback. The following playback technologies are supported:

- WebRTC
- MSE

Embedding page interface:

Embed player



Server ?

Stream ?

Tech WebRTC
 MSE

Code

```
<iframe id='fp_embed_player' src='http://localhost:9091/embed_player?urlServer=ws://localhost:8080&streamName=test&mediaProviders=WebRTC,MSE' marginwidth='0' marginheight='0' frameborder='0' width='100%' height='100%' scrolling='no' allowfullscreen='allowfullscreen'></iframe>
```

Code of the example

Source code of the example is on server by this path:

```
/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/embed\player
```

- `player.css` - CSS style file
- `player.html` - player page
- `player.js` - script for player to work
- `sample.css` - CSS style file for embedding interface page
- `sample.html` - embedding interface page

- `sample.js` - script to form embedding code

The example can be tested on this URL:

```
https://host:8888/client2/examples/demo/streaming/embed_player/sample.html
```

where `host` is your WCS server address

Analyzing the code

To analyze code get `player.js` file version with hash `24a69e1` that can be found [here](#) and is available to download in build [2.0.225](#).

1. API initializing

`Flashphoner.init()` [code](#)

```
Flashphoner.init({ preferredMediaProviders: mediaProviders && mediaProviders
  !== "" ? mediaProviders.split(',') : [] });
```

2. Connection to the server

`Flashphoner.createSession()` [code](#)

The following parameters are passed to `createSession()` method:

- `urlServer` - WCS server URL
- `mediaOptions` - parameters to connect through the [TURN server](#)

```
let mediaOptions = {"iceServers": [{url: 'turn:turn.flashphoner.com:443?
transport=tcp', 'username': 'flashphoner', 'credential':
'coM77EMrV7Cwhyah'}]};
Flashphoner.createSession({urlServer: urlServer, mediaOptions:
mediaOptions}).on(SESSION_STATUS.ESTABLISHED, function (session) {
  ...
});
```

3. Receiving the event confirming successful connection

`SESSION_STATUS.ESTABLISHED` [code](#)

```
Flashphoner.createSession({urlServer: urlServer, mediaOptions:
mediaOptions}).on(SESSION_STATUS.ESTABLISHED, function (session) {
  setStatus(session.status());
  //session connected, start playback
  playStream(session);
}).on(SESSION_STATUS.DISCONNECTED, function () {
```

```
    ...
  }).on(SESSION_STATUS.FAILED, function () {
    ...
  });
```

4. Video stream playback

`Session.createStream()`, `Stream.play()` [code](#)

The following parameters are passed to `createStream()` method:

- `streamName` - name of the stream
- `remoteVideo` - `div` element to display stream on page
- resolution to play the stream (transcoding will be enabled on server)
- `useControls` - enables a standard HTML5 video controls
- `unmutePlayOnStart: false` - disables automatic audio unmuting for autoplay to conform browsers requirements

```
let useVideoControls = true;
...
let options = {
  name: streamName,
  display: remoteVideo,
  useControls: useVideoControls
};
if (resolution) {
  playWidth = resolution.split("x")[0];
  playHeight = resolution.split("x")[1];
  options.constraints = {
    video: {
      width: playWidth,
      height: playHeight
    },
    audio: true
  };
}
if (autoplay) {
  options.unmutePlayOnStart = false;
}
playingStream = session.createStream(options).on(STREAM_STATUS.PENDING,
function (stream) {
  ...
});
playingStream.play();
```

5. Receiving the event confirming stream is ready to playback

`STREAM_STATUS.PENDING` [code](#)

On this event:

- hide the custom preloader in Chrome browser because there is a standard one when standard controls are enabled
- set up resize video event handler
- set up video event handlers separately for Safari and other browsers

```

playingStream = session.createStream(options).on(STREAM_STATUS.PENDING,
function (stream) {
    if (Browser.isChrome()) {
        // Hide a custom preloader in Chrome because there is a standard one
with standard controls
        hideItem('preloader');
    }
    let video = document.getElementById(stream.id());
    if (!video.hasListeners) {
        video.hasListeners = true;
        setResizeHandler(video, stream, playWidth);
        if (Browser.isSafariWebRTC()) {
            setWebkitEventHandlers(video);
        } else {
            setEventHandlers(video);
        }
    }
}).on(STREAM_STATUS.PLAYING, function (stream) {
    ...
}).on(STREAM_STATUS.STOPPED, function () {
    ...
}).on(STREAM_STATUS.FAILED, function(stream) {
    ...
}).on(STREAM_EVENT, function(streamEvent){
    ...
});
playingStream.play();

```

6. Receiving the event confirming successful stream playback

`STREAM_STATUS.PLAYING` code

On this event, MSE stream playback is unpaused in Android Firefox browser

```

playingStream = session.createStream(options).on(STREAM_STATUS.PENDING,
function (stream) {
    ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
    // Android Firefox may pause stream playback via MSE even if video
element is muted
    if (Flashphoner.getMediaProviders()[0] == "MSE" && autoplay &&
Browser.isAndroidFirefox()) {
        let video = document.getElementById(stream.id());
        if (video && video.paused) {
            video.play();
        }
    }
}
setStatus(STREAM_STATUS.PLAYING);

```

```

    onStart();
  }).on(STREAM_STATUS.STOPPED, function () {
    ...
  }).on(STREAM_STATUS.FAILED, function(stream) {
    ...
  }).on(STREAM_EVENT, function(streamEvent){
    ...
  });
  playingStream.play();

```

7. Stream playback stop

`Stream.stop()` [code](#)

```

playingStream.stop();

```

8. Receiving the event confirming successful playback stop

`STREAM_STATUS.STOPPED` [code](#)

```

playingStream = session.createStream(options).on(STREAM_STATUS.PENDING,
function (stream) {
  ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
  ...
}).on(STREAM_STATUS.STOPPED, function () {
  setStatus(STREAM_STATUS.STOPPED);
  onStop();
}).on(STREAM_STATUS.FAILED, function(stream) {
  ...
}).on(STREAM_EVENT, function(streamEvent){
  ...
});
playingStream.play();

```

9. Automatic playback starting if required

[code](#)

```

if (autoplay) {
  centralButton.click();
}

```

10. Setting up `resize` event handler

[code](#)

On this event, the container size for video element is changed

```

function setResizeHandler(video, stream, playWidth) {
  video.addEventListener('resize', function (event) {
    let streamResolution = stream.videoResolution();
    if (Object.keys(streamResolution).length === 0) {
      resizeVideo(event.target);
    } else {
      // Change aspect ratio to prevent video stretching
      let ratio = streamResolution.width / streamResolution.height;
      let newHeight = Math.floor(playWidth / ratio);
      resizeVideo(event.target, playWidth, newHeight);
    }
  });
}

```

11. Setting up event handlers for Safari browser

code

The following events are handled:

- `playing` - hide the custom preloader when stream is playing
- `webkitbeginfullscreen`, `webkitendfullscreen` - detect full screen mode to unpause stream playback when exiting this mode in iOS Safari
- `pause` - unpause stream playback when exiting full screen mode; stop playback by clicking the standard pause control in windowed mode

```

function setWebkitEventHandlers(video) {
  let needRestart = false;
  let isFullscreen = false;
  // Hide custom preloader
  video.addEventListener('playing', function () {
    hideItem('preloader');
  });
  // Use webkitbeginfullscreen event to detect full screen mode in iOS
  Safari
  video.addEventListener("webkitbeginfullscreen", function () {
    isFullscreen = true;
  });
  video.addEventListener("pause", function () {
    if (needRestart) {
      console.log("Video paused after fullscreen, continue...");
      video.play();
      needRestart = false;
    } else if (!(isFullscreen || document.webkitFullscreenElement)) {
      // Stop stream by standard play/pause control
      playingStream.stop();
    }
  });
  video.addEventListener("webkitendfullscreen", function () {
    video.play();
    needRestart = true;
    isFullscreen = false;
  });
}

```

```
});  
}
```

12. Setting up event handlers in other browsers

code

The following events are handled:

- `playing` - hide the custom preloader when stream is playing
- `pause` - stop playback by clicking the standard pause control in windowed mode

```
function setEventHandlers(video) {  
  // Hide custom preloader  
  video.addEventListener('playing', function () {  
    hideItem('preloader');  
  });  
  // Use standard pause control to stop playback  
  video.addEventListener("pause", function () {  
    if (!(document.fullscreenElement || document.mozFullscreenElement)) {  
      // Stop stream by standard play/pause control if we're not in  
      fullscreen  
      playingStream.stop();  
    }  
  });  
}
```