WebRTC_call

API allows for establish audio/video call between two **BROWSERS** or between **BROWSER** and **SIP CLIENT**.

Before establishing the call it is necessary to **REGISTER** in Orange IMS network using API. To start build new services developers MUST authenticate using credential provided by Orange.

All calls established based on this API will be provided using Orange network and infrastructure.

Test phones number will be delivered by Orange. These phones will be registered in Orange IMS network and can be used to test service.

Authorization

Before the start of application development it is necessary to authorize to Webrtc Session Controller using Restful Authentication using correct credentials delivered by Orange. Only after successful authorization the registeration in Orange IMS network is possible.

```
function loginToWsc() {var wscDemoBaseUrl =
"http://host:port/demo.html";
     window.location.href =
"http://webrtc.bihapi.pl:7001/login?
     wsc_app_uri=ws/webrtc/restauth&redirect_uri=" +
wscDemoBaseUrl + "?auth=secure";
}
<!doctype html>
<html>
     <head>
          <title>Demo</title>
     </head>
     <script>
          function loginToWsc() {
               var wscDemoBaseUrl =
"http://host:port/demo.html";
               window.location.href =
"http://webrtc.bihapi.pl:7001/login?
     wsc_app_uri=ws/webrtc/restauth&redirect_uri=" +
wscDemoBaseUrl +            "?auth=secure";
          }
     </script>
     <body>
          <h4>Login to WebRTC Session Controller</h4>
          <a href="#" onclick="loginToWsc()">Login</a>
```

```
    </body>
</html>
```

- **wscDemoBaseUrl**: url for developer created application
- **wsc_app_uri=http://webrtc.bihapi.pl:7001/ws/webrtc/restauth** - allows for login to Webrtc Session Controller using Restful Authentication.

After successful login user will be redirected to wscDemoBaseUrl. All necessary credentials required to register in Orange IMS network are provided automatically. You can start develop application.

To start using API you MUST include specific file "wsc.js"

<script src="http://webrtc.bihapi.pl:7001/api/wsc.js" type="text/javascript"></script>

Managing session with wsc.Session

The managing the WebSocket connection between the application and WebRTC Session Controller provides the following information when you create your application's session object:

username: The user's name

wsUri: The WebSocket URI

onsessionSuccess: The callback function in your application that should be invoked when the session is created

onSessionError: The callback function in your application that should be invoked when there is an error in creating the session

```
var wscSession;
var wsUri = "ws://" +
"webrtc.bihapi.pl:7001/ws/webrtc/restauth";
wscSession = new wsc.Session(userName, wsUri,
onSessionSuccess, onSessionError);
function onSessionSuccess() {
    setStatus('onSessionSuccess: userName ' +
wscSession.getUserName());
    //make specific action
}

function onSessionError() {
    setStatus('onSessionError');
    /make specific action
}
```

Handling Session State Changes

Session state values are constants, such as **CLOSED** or **CONNECTED**. Session states are defined in the **wsc.SESSIONSTATE** enumerator.

If wscSession is your application's session object, you can set up a callback function and assign that function to your application's Session.onSessionStateChange event handler. Whenever the state of your application's session changes, the WebRTC Session Controller JavaScript API library invokes your application's Session.onSessionStateChange event handler and provides the new state.

In the callback function, you can check the new session state against the defined constants and set up appropriate actions to respond to the new state. For example, a change in the value of wsc.SESSIONSTATE from **RECONNECTING** to **CONNECTED** indicates that the attempt to reconnect succeeded and that the application can proceed. Or if the state changes from **RECONNECTING** to **FAILED**, the attempt to reconnect failed. In each case, your application may need to take appropriate action with respect to the user.

```
wscSession.onSessionStateChange = onSessionStateChange;
function onSessionStateChange(sessionState) {
      switch (sessionState) {
            case wsc.SESSIONSTATE.RECONNECTING :
                  //make specific action
                  break;

            case wsc.SESSIONSTATE.CONNECTED :
                  //make specific action
                  break;
            case wsc.SESSIONSTATE.FAILED :
                  //make specific action
                  break;
      }
}
```

Managing Calls with wsc.CallPackage

Class to manage audio or video communication and/or data transfers in calls made from or received by your application.

After creating an instance of the CallPackage class in your application, assign a callback function to handle each of the following events:

```
callPackage = new wsc.CallPackage(wscSession)
     if (callPackage) {
```

```
          callPackage.onIncomingCall = function(callobj,
callConfig) {
               onIncomingCall(callobj, callConfig)
          };
          callPackage.onResurrect = onResurrect;
     }
function onIncomingCall(callobj, remoteCallConfig) {
     var callType = "audio:" +
remoteCallConfig.audioConfig + ",video:" +
     remoteCallConfig.videoConfig;
     document.getElementById('accept').onclick =
function() {
          callobj.accept(callobj.callConfig);
     };
     document.getElementById('reject').onclick =
function() {
          callobj.decline();
     };
}
function onResurrect(resurrectedCall) {
     //make specific action
}
```

An incoming call, using the **onIncomingCall** event handler.

In this callback function, implement the logic to process the incoming call, such as filtering to reject calls from blacklisted numbers or responding when the user accepts or declines the call.

A reconnected call, using the **onResurrect** event handler.

In this callback function, implement the logic to handle the call that was dropped momentarily.

Managing a Call with wsc.Call

When the application accepts an incoming call, use the incoming call object and the remote call configuration for the resulting call session.

Manage changes in the state of the call and its associated audio, media or data channel with the event handlers of the wsc.Call class, by implementing the logic in the callback function you assign for each event.

For Call state changes, use the onCallStateChange event handler. Your application needs to respond to the changes in the state of a call. The fields of the

wsc.CALLSTATE enumerator object hold the various states of a call, such as **STARTED, RESPONSED, and ENDED**.

```
callobj.onCallStateChange = function(newState) {
          onCallStateChange(callobj, newState);
     };
function onCallStateChange(callobj, callState) {
     if (callState.state == wsc.CALLSTATE.ESTABLISHED) {
          //make specific action
     }
     else if (callState.state == wsc.CALLSTATE.ENDED){
          //make specific action
     }else if (callState.state == wsc.CALLSTATE.FAILED) {
          //make specific action
     }
}
```

Media state changes, use the onMediaStreamEvent event handler. Your application needs to respond to changes in the media stream states of the call, whether it is voice or video. The WebRTC Session Controller JavaScript API library provides the wsc.MEDIASTREAMEVENT enumerator which defines the following three states each for the local and remote streams.

Added (**LOCAL_STREAM_ADDED or REMOTE_STREAM_ADDED**)

Removed (**LOCAL_STREAM_REMOVED or REMOTE_STREAM_REMOVED**)

In error (**LOCAL_STREAM_ERROR or REMOTE_STREAM_ERROR**)

```
callobj.onMediaStreamEvent = mediaStateCallback;
function mediaStateCallback(mediaState, stream) {
     if (mediaState ==
wsc.MEDIASTREAMEVENT.LOCAL_STREAM_ADDED) {
          setStatus("Local video stream is added...");
     attachMediaStream(document.getElementById("selfView")
, stream);
     } else if (mediaState ==
wsc.MEDIASTREAMEVENT.REMOTE_STREAM_ADDED) {
          setStatus("Remote video streama is added...");
     attachMediaStream(document.getElementById("remoteView
"), stream);
     }
}
```

Specifying the Configuration for Calls with wsc.CallConfig

Specify the audio, video, and data channel capability for calls made from your application, with the wsc.CallConfig class.

When you create an instance of the wsc.CallConfig class in your application, set up the direction of the audio and video elements in the local media stream. Use the **wsc.MEDIADIRECTION**: enumerator to specify the direction of the local media stream as one of the following:

wsc.MEDIADIRECTION.SENDRECV: which indicates that the local media stream can send and receive the media stream.

wsc.MEDIADIRECTION.SENDONLY: which indicates that the local media stream can send the media stream.

wsc.MEDIADIRECTION.RECVONLY: which indicates that the local media stream can receive the media stream.

wsc.MEDIADIRECTION.NONE: which indicates that media is not supported.

```
var audioMediaDirection = wsc.MEDIADIRECTION.SENDRECV;
var videoMediaDirection = wsc.MEDIADIRECTION.SENDRECV;
callConfig = new wsc.CallConfig(audioMediaDirection,
videoMediaDirection);
var callee = '+48399xxxxxx@neofon.tp.pl';
var call = callPackage.createCall(callee, callConfig,
doCallError);
if (call != null) {
     call.onCallStateChange = function(newState) {
          onCallStateChange(call, newState);
     };
     call.onMediaStreamEvent = mediaStateCallback;
     call.start();
}
```

Atach Media Stream

```
var attachMediaStream = null;
var reattachMediaStream = null;
var webrtcDetectedBrowser = null;
if (navigator.mozGetUserMedia) {
      webrtcDetectedBrowser = "firefox";
      // Attach a media stream to an element.
      attachMediaStream = function(element, stream) {
            console.log("Attaching media stream");
            element.mozSrcObject = stream;
            element.play();
      };
      reattachMediaStream = function(to, from) {
            console.log("Reattaching media stream");
            to.mozSrcObject = from.mozSrcObject;
```

```
            to.play();
      };
} else if (navigator.webkitGetUserMedia) {
      webrtcDetectedBrowser = "chrome";
      // Attach a media stream to an element.
      attachMediaStream = function(element, stream) {
            element.src =
webkitURL.createObjectURL(stream);
      };
      reattachMediaStream = function(to, from) {
            to.src = from.src;
      };
      // The representation of tracks in a stream is
changed in M26.
      // Unify them for earlier Chrome versions in the
coexisting period.
      if (!webkitMediaStream.prototype.getVideoTracks) {
            webkitMediaStream.prototype.getVideoTracks =
function() {
                  return this.videoTracks;
            };
            webkitMediaStream.prototype.getAudioTracks =
function() {
                  return this.audioTracks;
            };
      }
      // New syntax of getXXXStreams method in M26.
      if
(!webkitRTCPeerConnection.prototype.getLocalStreams) {

      webkitRTCPeerConnection.prototype.getLocalStreams =
function() {
                  return this.localStreams;
            };
      webkitRTCPeerConnection.prototype.getRemoteStreams
= function() {
                  return this.remoteStreams;
            };
      }
} else {
      //console.log("Browser does not appear to be
WebRTC-capable");
}
```

Examples/Testing

Service developed by you can be launch only on WebRTC capable web browser. Orange recommends using latest version of Chrome or Firefox web browser.

Example 1: WebRTC_call - Web 2 Web scenario

On first web browser launch your service and register using first phone number delivered to you by Orange

On second web browser launch your service and register using second number delivered to you by Orange

Call from one number to second number using audio/video calls. To start call you have to use correct form **phone@neofon.tp.pl** where **phone** is a number provided by Orange e.g. +48399500076@neofon.tp.pl

All calls required user permission to use microphone or camera.

Example 2: WebRTC_call - Web 2 SIP scenario

Built environment allows you to make a call in web to sip scenario. You can make an audio/video call directly from web browser to sip client or from sip client to web browser (**in case of video calls the sip client must support VP8 video codec**) Orange recommends using CounterPath Bria 4 SIP Client. The desktop version of this client was tested by Orange for audio and video calls as well. The client allows making audio and video call in web 2 sip scenario. You can use any other SIP client but Orange can't ensure that this client will be compatible with built environment.

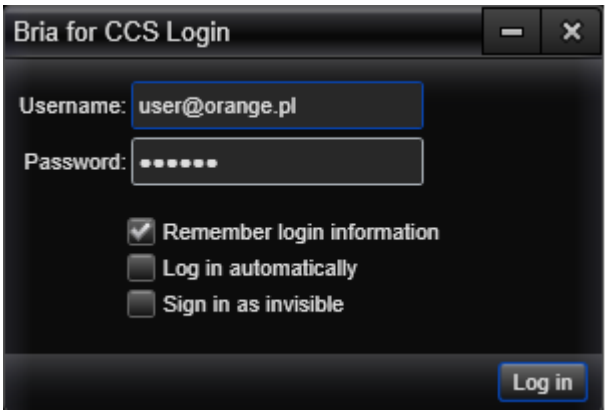
To test service in case of web to sip scenario you can use CounterPath Bria4 Client recommend by Orange. The license for this client is valid during the contest. To get the free version of client you have to:

1. Download the client
    a. Mobile/tablet - Go to store on your device and search Bria client. Choose free (RED) Bria SaaS client
    b. Windows platform – download the client from

http://download.counterpath.com/dl.php?RemoteID=y1RE0c9qstyMvnirnPlv9tpAs

c. Mac platform – download the client from http://download.counterpath.com/dl.php?RemoteID=ifu5rewazh1e2fafygxxbgl3

2. Configure client using credentials send to you (in dedicated mail) by Orange. The Bria4 client will be automatically configured and ready to use



Please provide:

- Username provided to you by Orange e.g. name@orange.pl
- Password provided to you by Orange e.g. "password"

Click login, after few second the client will be registered in Orange network and ready to use.

On web browser register the second number deliver by Orange.
You can call from web browser to CounterPath Bria4 client using audio and video call. You can call from CounterPath Bria 4 client to web browser using audio and video call as well.

Remarks

1. In case of **AUDIO** calls in **Web2SIP** scenario Orange recommend to use **G711 aLaw** or **G711 uLaw** audio codec (you can configure it on Sip Client)
2. All video calls are done based on **VP8** video codec. Orange highly recommends using **licensed desktop version of CounterPath Bria4** SIP client (www.counterpath.com/bria). You can use any SIP client which supports VP8 video codec but Orange can't ensure that this client will be compatible with built environment.
3. The quality of video call is depends on many factors as Internet bandwidth, environment parameters (CPU, RAM, etc.), camera quantity
4. Created service can be launched only on **WebRTC capable browser**. Orange recommends using latest version of Chrome or Firefox browser.
5. WebRTC uses microphone and/or camera. The permission to use it is required from user.
6. This document is only abstract describing the most basic functionality of Oracle Communication WebRTC Session Controller (OCWSC). To get more information about other features please read full documentation available on Internet.
7. In case of problem with not displaying remote video stream form Bria Client on web browser please try to stop sending stream and start it again