open·tok

Tutorials and Sample Code

JavaScript API: Tutorials and Sample Code

Basic Functionality

Hello World Tutorial

Shows you how to connect to a session, publish a stream from your webcam, and view other audio-video streams in the session … all in just a few lines of code.

Basic Tutorial

Expands on the Hello World tutorial, showing how to position OpenTok videos in the HTML DOM.

Advanced Tutorials

Controlling audio

Shows how to enable echo suppression, how to mute the microphone and how to mute sound output from streams.

Managing cameras and microphones

Shows how to use JavaScript to have the user select the camera and microphone used in an OpenTok session.

Moderation and signaling

Shows how to moderate participants in an OpenTok session, and how to have a participant send a signal to all other participants.

Targeting mobile platforms

Shows how to use "auto-produced" streams. An auto-produced stream is one that multiplexes many all of the audio-video streams for a session into a single stream. This stream automatically switches the video to display the the active participant in the session (based on audio). This tutorial also shows how to check if a user has the capability to publish videos to an OpenTok session.

Resizing videos

Shows how you can control the size of the OpenTok video display, just as you would any other HTML element.

Audio-only and video only

Shows how to publish streams and subscribe to streams in audio-only and video-only mode.

JavaScript Tutorial — Hello World Tutorial

This tutorial shows how to connect to an OpenTok™ session, display any existing streams and publish a video stream to the session. Although this is not a traditional "hello world" sample, it does show the most basic application to use the most basic OpenTok functionality.

Testing the tutorial online

To run the tutorial at the top of this page:

1. Make sure that you have a webcam connected to your computer and configured to run.
2. In the example at the top of this page, if the **Flash Player Settings** dialog is displayed, click the **Allow** button. This grants the application access to your camera and microphone.

   The example now connects to a sample OpenTok session. It displays any other video streams in the session (if there are any), and it displays and publishes your video to the session.
3. Mute the speaker on your computer. (For this test app, this will prevent audio feedback.)
4. Copy the URL for this page into the Clipboard.
5. Open a new browser window, and have it open to the copied URL.

   For test purposes, you can view the page in the new browser window on your computer. Or you can open it on another computer (as would happen in a real session.)

   The new page now connects to the session. Upon connection the video stream from the other page is displayed on the page and the new published video stream appears on both pages.

   You can wave to the camera and say "hello world."

Understanding the code

This application shows the simplest example of connecting to an OpenTok session, displaying video streams existing in the session, and publishing a video stream to a session.

We assume you know how to create HTML Web pages and have a basic understanding of JavaScript and the HTML Document Object Model. If you are not comfortable with those technologies, then you should consider using the TokBox Embed option instead.

Adding the OpenTok JavaScript library

To add the OpenTok JavaScript library, the <head> section of the webpage includes the following line:

```
<script src="http://staging.tokbox.com/v0.91/js/TB.min.js" ></script>
```

This one script includes the entire OpenTok JavaScript library. The `src` URL will change with future versions of the OpenTok API. However, TokBox™ will maintain this version at the original URL.

Connecting to an OpenTok session

The first step in using an OpenTok session is to create and initialize a Session object. The Session object is defined in the OpenTok JavaScript library. You initialize a Session object by calling the `TB.initSession()` method with the session ID pertaining to the OpenTok session. In this case, it is the ID for the sample session used for this demo:

```
var session = TB.initSession("1sdemo00855f8290f8efa648d9347d718f7e06fd");
```

The TB object and Session object are defined in the OpenTok JavaScript library. (See the OpenTok JavaScript API reference for details.)

Once the initialization is complete, you can connect to the session by calling the `connect()` method of the Session object. The `connect()` method takes two arguments: the API key and the token string:

```
session.connect(1127, "devtoken");
```

The API key identifies you as an OpenTok developer. (In this case, it is a sample API key used for the tutorial.) The token string defines a user. The Basic Tutorial describes these in more detail.

Before calling the `connect()` method of the Session object, the code adds event listeners that enable the OpenTok controller to send events to JavaScript functions:

```
session.addEventListener("sessionConnected", sessionConnectedHandler);
session.addEventListener("streamCreated", streamCreatedHandler);
```

The `sessionConnectedHandler()` function calls a function that subscribes to the existing streams in the session, and it publishes your camera's video to the session:

```
function sessionConnectedHandler (event) {
     subscribeToStreams(event.streams);
     session.publish();
}

function subscribeToStreams(streams) {
    for (i = 0; i < streams.length; i++) {
        var stream = streams[i];
        if (stream.connection.connectionId != session.connection.connectionId) {
            session.subscribe(stream);
        }
    }
}
```

The `publish()` method of a Session object lets you publish your webcam's audio-video stream to the session.

The `streamCreatedHandler()` function processes streams that are added to a session (after you initially connect):

```
function streamCreatedHandler(event) {
    subscribeToStreams(event.streams);
}
```

Both the `sessionConnected` event and the `streamCreated` event objects contain a `streams` property, which is an array of streams. The `subscribeToStreams()` function calls the `subscribe()` method of the OpenTok Session object to subscribe to each stream.

Note that the `subscribeToStreams()` function checks to make sure that a stream does not correspond to the one you are publishing (as determined by the `session.connection.connectionId` property).

When the code calls `publish()` and `subscribe()`, this simple sample app simply adds video streams in new DIV elements appended to the HTML body. However, you can (and usually will) want to assign streams to HTML elements you define. The `publish()` and `subscribe()` methods each take an optional `replaceElementId` parameter. To see how this works, look at the Basic Tutorial.

Testing the code on your own computer

You can download the source code and test it on your own computer:

1. Click the **Download the source code** link at the top of this page (just beneath the example).
2. Extract the ZIP file you download, and open the helloworld.html file in a web browser.

If the web page asks you to set the Flash Player Settings, or if you do not see a display of your camera in the page, do the following:

1. Open this link in a new browser window: http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html .

   The Flash Player Global Security Settings panel is displayed.

2. In the **Alway trust files in these locations** list click the **Edit locations** link and select **Add location**.
3. Click **Browse for folder**, navigate to the folder that contains the files you downloaded, and then click the **Open** button.
4. Click the **Always Allow** button.

   This grants the HTML pages in the folder you selected to communicate with Flash content served by tokbox.com. (It also grants this communication from this folder to other servers, so be careful to select a specific folder.)

5. Reload the local version of the HTML file in your web browser.

Next steps

Now that you have looked at this simple example, go ahead and read the Basic Tutorial. Then look at the other tutorials to learn more of the OpenTok functionality.

You may also want to look at the OpenTok documentation.

JavaScript Tutorial — Basic Tutorial

This tutorial shows how to connect to an OpenTok™ session, display any existing streams and publish a video stream to the session.

Testing the tutorial online

To run the tutorial at the top of this page:

1. Make sure that you have a webcam connected to your computer and configured to run.
2. Click the **Publish** link, at the top of the page.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button. This grants the application access to your camera and microphone.

   You are now publishing a video stream to the session, and the video stream is displayed on the page.
3. Copy the URL for this page into the Clipboard.
4. Open a new browser window, and have it open to the copied URL.

   For test purposes, you can the new browser window on your computer. Or you can open it on another computer (as would happen in a real session.)
5. Click the **Connect** link, at the top of the page (in the new browser window).

   The new page now connects to the session. Upon connection the video stream from the other page is displayed on the page.
6. Click the **Publish** link, at the top of the page.

   You are now publishing a new video stream to the session. This new video stream is displayed on the other browser page.
7. Click the **Unpublish** link.

   The page removes the video from the session. It is also removed from all pages.
8. Click the **Disconnect** link.

   This disconnects you from the session (and removes all videos from the page).

Understanding the code

We assume you know how to create HTML Web pages and have a basic understanding of JavaScript and the HTML Document Object Model. If you are not comfortable with those technologies, then you should consider using the TokBox Embed option instead.

Adding the OpenTok JavaScript library

To add the OpenTok JavaScript library, the <head> section of the webpage includes the following line:

```
<script src="http://staging.tokbox.com/v0.91/js/TB.min.js" ></script>
```

This one script includes the entire OpenTok JavaScript library. The `src` URL will change with future versions of the OpenTok API. However, TokBox™ will maintain this version at the original URL.

Connecting to an OpenTok session

The top of the code defines these values, which identify the developer, the OpenTok session and the user:

```
<script type="text/javascript">

var apiKey = 1127; // OpenTok sample code key. Replace with your own API key.
var sessionId = '153975e9d3ecce1d11baddd2c9d8d3c9d147df18'; // Replace with your
session ID.
var token = 'devtoken'; // Should not be hard-coded.
                         // Add to the page using the OpenTok server-side libraries.
```

These variables define the following values:

- The *API key* identifies you as a developer registered to use the OpenTok API. You will obtain an API key from TokBox when you register to use the OpenTok API (at the Get your API key page). While testing out the basic tutorial application, you can use the sample code API key (1127). When you develop your own application, you will use the API key assigned to you.
- The *session ID* is a unique identifier for your OpenTok video chat session. For this basic tutorial application, you can use the provided session ID. For your own application, you will obtain your own session ID from TokBox.
- The *token* string identifies a user. While using a test version of an application (such as this basic tutorial), you can use the `"devtoken"` token string. When you deploy your application, you will use the OpenTok server-side libraries to obtain a unique token string for each participant in the session. This token string is used to validate a participant.

The first step in using an OpenTok session is to create and initialize a Session object. The Session object is defined in the OpenTok JavaScript library. You initialize a Session object by calling the `TB.initSession()` method with the session ID pertaining to the OpenTok session:

```
session = TB.initSession(sessionId);
```

The TB object and Session object are defined in the OpenTok JavaScript library.

Once the initialization is complete, you can connect to the session by calling the `connect()` method of the Session object. The `connect()` method takes two arguments: the API key and the token string:

```
var session;
session = TB.initSession(sessionId);

function connect() {
    session.connect(apiKey, token);
}
```

Note: The objects, methods, properties, and events in the OpenTok JavaScript API are described in the OpenTok JavaScript API reference.

In the basic tutorial application, the **Connect** link in the HTML page calls the `connect()` function (which calls the `connect()` method of the Session object).

Before calling the `connect()` method of the Session object, the code adds event listeners that enable the OpenTok controller to send events to JavaScript functions:

```
var session;
var publisher;
var subscribers = {};

// TB.setLogLevel(TB.DEBUG);
if (TB.checkSystemRequirements() != TB.HAS_REQUIREMENTS) {
    alert('Minimum System Requirements not met!');
}

TB.addEventListener('exception', exceptionHandler);
session = TB.initSession(sessionId);

session.addEventListener('sessionConnected', sessionConnectedHandler);
session.addEventListener('sessionDisconnected', sessionDisconnectedHandler);
session.addEventListener('connectionCreated', connectionCreatedHandler);
session.addEventListener('connectionDestroyed', connectionDestroyedHandler);
session.addEventListener('streamCreated', streamCreatedHandler);
session.addEventListener('streamDestroyed', streamDestroyedHandler);

function connect() {
    session.connect(apiKey, token);
}
```

Objects in the OpenTok JavaScript library uses the `addEventListener()` method to register event listener function to events. Event types are identified with strings. For example, the Session object dispatches a `sessionConnected` event when the OpenTok controller successfully connects to the OpenTok session. You can add the listeners for all the events you are interested in subscribing to.

The sections that follow will describe the event listeners in this code.

Publishing an audio-video stream to an OpenTok session

The `publish()` method of a Session object lets you publish your webcam's audio-video stream to the session:

```
var div = document.createElement('div');
div.setAttribute('id', 'publisher');
document.body.appendChild(div);
publisher = session.publish('publisher');
publisher.addEventListener('settingsButtonClick', showSettings);
```

The `publish()` takes the ID of an HTML element as the first parameter (which in this case is a `div` element with the ID "publisher"). The `publish()` method replaces with a video box (publisher) containing audio and video.

The `publish()` method also returns a publisher object, which can be used to perform operations (like unpublish the stream if you want to). In this case we keep track of the publisher object so we can use it to unpublish or perform custom clean up later on.

The `unpublish()` method of the Session object removes your published video stream from the session. It also removes the display of the video from the page. The **Unpublish** link on the page calls the `unpublish()` function:

```
if (publisher) {
    session.unpublish(publisher);
}
publisher = null;
```

Subscribing to streams

When users publish streams to an OpenTok session, the Session object dispatches a StreamConnected event. It also dispatches this event when you first connect to the session. The code in this tutorial registers the `streamCreatedHandler()` method as the event listener for the `streamCreated` event:

```
function streamCreatedHandler(event) {
    for (var i = 0; i < event.streams.length; i++) {
        addStream(event.streams[i]);
    }
}
```

The `streamCreated` event object is defined by the StreamEvent object in the OpenTok JavaScript library. The StreamEvent object includes a `streams` property, which is an array of Stream objects pertaining to the event. The code above loops through all the streams available and calls the `addStream()` function:

```
function addStream(stream) {
    if (stream.connection.connectionId == session.connection.connectionId) {
        return;
    }
    var div = document.createElement('div');
    var divId = stream.streamId;
    div.setAttribute('id', divId);
    document.body.appendChild(div);
    subscribers[stream.streamId] = session.subscribe(stream, divId);
}
```

The `addStream()` function takes a `stream` object and subscribes to it. The code first creates an HTML `div` element which will be replaced by the `subscriber` UI component. The `subscribe()` method of the Session object takes two arguments: a Stream object and the HTML element. In the first line of the function above, the code checks to see if a Stream object references the stream we published. We also use the `stream.streamid` as the `div` ID, so we have an easy, mechanism to get unique IDs. In this case we also keep track of all `Subscriber` objects so we can use them for cleanup at some later point in time.

The Session object dispatches a `streamDestroyed` event when a stream leaves a session. This may happen if a user disconnects or stops publishing a stream. The default behavior for this event is to unsubscribe from any Subscribers corresponding to the stream. (This default behavior was added in OpenTok v0.91.) You can cancel the default behavior by calling the `preventDefault()` method of the `streamDestroyed` event object. If you cancel the default behavior, you can remove Subscriber objects using your own code by calling the `unsubscribe()` method of a Subscriber object. (The `streams` property of the event is an array of Stream objects. The `getSubscribersForStream()` method of the Session object lists the Subscriber objects pertaining to a Stream.)

The Subscriber object is defined by the OpenTok JavaScript library. You created a Subscriber object when you subscribed to each stream, using the `subscribe()` method of the Session objects. You can subscribe to a stream multiple times (putting it in multiple places on the HTML page). However this tutorial only subscribes to each stream once.

Note also that the `streamCreated` and `streamDestroyed` events each include a `streams` property. These may contain only one Stream object (when a single stream is added or removed from a session).

Disconnecting from the session

When the user clicks the **Disconnect** link on the page, the `disconnect()` function is called. This in turn calls the `disconnect()` method of the Session object. Disconnecting from a session is an asynchronous operation. When the session has disconnected, the Session object dispatches the `sessionDisconnected` event, which is handled by the the `sessionDisconnected()` function. This function simply changes the page to show the Connect link.

The default behavior of the `sessionDisconnected` event is to unsubscribe from any Subscribers corresponding to the stream and remove the Publisher from the page. (This default behavior was added in OpenTok v0.91.) You can cancel the default behavior by calling the `preventDefault()` method of the `sessionDisconnected` event object. If you cancel the default behavior, you can remove Subscriber and Publisher objects using your own code. You can call the `unsubscribe()` method and the `unpublish()` method of the Session object. (The Session object has a `subscribers` property and a `publishers` property, which are arrays of Subscriber and Publisher objects.)

The Session object also dispatches a `sessionDisconnected` event if a session connection is lost inadvertantly, as in the case of a lost network connection.

Handling errors and debugging

The TB object dispatches `exception` event when the application encounters an error event.

The `exceptionHandler()` function is registered as the event listener for error events:

```
function exceptionHandler(event) {
    alert("Exception msg:" + event.message +
            "title: " + event.title + " code: " + event.code);
}
```

However, the event handler for the `exception` event is commented out near the top of the code (in the second line shown here):

```
// TB.setLogLevel(TB.DEBUG);
// TB.addEventListener("exception", exceptionHandler);
```

You can uncomment the call to the `addEventListener()` method of the TB object to register the event listener. Or you can uncomment the first line to turn on the OpenTok logging feature. When you call `TB.setLogLevel(TB.DEBUG)`, the OpenTok JavaScript library logs API actions to a debugger console, such as FireBug. Included in the logged messages are warnings, errors, and other useful information. If no debugger is connected, errors are displayed in JavaScript alert messages.

Note that the code also checks to see if the OpenTok API is supported on the client browser:

```
if (TB.checkSystemRequirements() != TB.HAS_REQUIREMENTS) {
    alert('Minimum System Requirements not met!');
}
```

Testing the code on your own computer

See the instructions in the Hello World tutorial.

Creating a production version of an application

Once you are ready to show your app to the world, do the following:

1. Register your app with Tokbox. See the Launch page.
2. Use one of the OpenTok server-side libraries to obtain session IDs and unique token strings for each user. Make sure that the `API_URL` constant in the library you use is set to `http://staging.tokbox.com/hl`. See the OpenTok server-side libraries documentation.
3. Load the OpenTok JavaScript library from the following URL: `<script src="http://staging.tokbox.com/v0.91/js/TB.min.js"></script>`.

For more information on testing and production versions of applications, see Testing and Production.

JavaScript Tutorial — Controlling Audio

This tutorial shows how to use the TokBox API to control audio in a session:

- You can enable echo suppression.
- You can control the streaming of and subscribing to audio for all streams to enable push-to-talk functionality

To see a tutorial on how to publish or subscribe to a stream with audio or video only, see the Audio-only and Video-only tutorial.

Testing the tutorial online

To run the tutorial:

1. Make sure that you have a webcam connected to your computer and configured to run, and that you are wearing headphones.
2. Click the **Start Publishing** link.
3. If a JavaScript alert message is displayed (about headphones and push-to-talk), click the **Cancel** button (even if you are wearing headphones).

   The page does not display this message if the last user of the tutorial left it in echo-suppression mode.

   Clicking the **Cancel** button, puts the session in echo-suppression mode.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button and then click the Close button. This grants TokBox.com access to your camera and microphone.

   You are now publishing a video stream to the session, and the video stream is displayed on the page.
4. Copy the URL for this page into the Clipboard.
5. Open a new browser window, and have it open to the copied URL.

   Better yet ... have another user open the URL in a browser on another computer.

   The new page now connects to the session. Upon connection the video stream from the other page is displayed on the page.
6. Click the **Start Publishing** link (in the new browser window).

   The new page now publishes a new video stream to the session. Now both pages display both streams.
7. Click the **Disable echo suppression** link. Then, in the alert message that is displayed, click the **OK** button (even if you are wearing headphones).

   Echo suppression is now disabled for the session. Each user must now press the **Click**

to talk button to enable their microphone.

8. Click the **Click to talk** button and speak.

   While in this mode, all other streams (from other pages) are no longer subscribed to locally. This prevents any possibility of echo feedback.

9. Click the **Click to mute** button.

   Your microphone is now muted.

Testing the code on your own computer

See the instructions in the Hello World tutorial.

Understanding the code

Note: You may want to look at the Basic Tutorial sample, if you have not already. It provides details on many of the basic concepts, like connecting to an OpenTok session, publishing an audio-video stream, and subscribing to streams.

To understand the OpenTok JavaScript library is used, View the source for this browser page. You can also download the source for this Tutorial application (and for the other OpenTok tutorial applications) by clicking the **Download Source** button at the top of this page.

Enabling echo suppression

When you click the **Enable echo suppression** button, the `onclick` handler for the button calls the following code:

```
globalGroup.enableEchoSuppression()
```

The `globalGroup` variable was set by the event handler for the `sessionConnected` event:

```
function sessionConnectedHandler(event) {

    // ...

    globalGroup = event.groups[0];

    // ...
}
```

The Session object dispatches a `sessionConnected` event when the connection to the OpenTok session is established. The event dispatched is a SessionConnectEvent object. This object has a `groups` property, that is an array of Group objects. A Group object references an OpenTok group. In this version of the OpenTok API, each session has exactly one group (the global group), corresponding to the first element of the `groups` array (`groups[0]`).

The `enableEchoSuppression()` method of this Group object is called when you press the **Enable echo suppression** button. This method enables echo suppression for the group.

Echo suppression is applied to individual streams only. With echo suppression set, only one individual stream (the current participant) will have audio output at one time. Echo suppression does not affect the audio output of multiplexed streams.

Note: For information on code that initiates a TokBox session, connects to the session, see the Basic tutorial.

When you enable echo suppression, the Group object dispatches a `groupPropertiesUpdated` event. The code on the page includes a `groupPropertiesUpdated()` function which is the event listener for the `groupPropertiesUpdated` event:

```
function groupPropertiesUpdatedHandler(event) {
    if (event.target.groupId == globalGroup.groupId) {
        if (event.target.getGroupProperties().echoSuppression.isEnabled) {
            setEchoSuppressionButton("disable");
        } else {
            setEchoSuppressionButton("enable");
            if (publisher) {
                // Mute audio to remove risk of feedback
                publisher.publishAudio(false);
            }
        }
    }
}
```

Among other things, the `groupPropertiesUpdatedHandler()` function toggles between displaying the **Enable echo suppression** button and the **Disable echo suppression** button.

Note: This is an abbreviated version of the function. Other code in the function deals with push-to-talk functionality and with muting and unmuting microphones and streams. These are discussed in the next sections.

When the user clicks the **Disable echo suppression** button, the button's `onclick` handler calls the following code:

```
globalGroup.disableEchoSuppression()
```

This calls the `disableEchoSuppression()` method of the global Group object. This disables echo suppression for the group.

Note that when you toggle between the **Enable echo suppression** and **Disable echo suppression** buttons, the buttons toggle on all pages connected to the session. That is because the Group object on all pages dispatches the `groupPropertiesUpdated` event, and the event handler toggles the button state.

For more information on OpenTok groups, see the Auto-production sample application.

Publishing your audio and subscribing to other's audio

The OpenTok JavaScript library includes the following methods for publishing your audio stream and connecting to the audio streams of other participants in the session:

- The `publishAudio()` method of the Publisher object
- The `subscribeToAudio()` method of the Subscriber object

The **Click to talk** and **Click to mute** buttons toggle audio input and output:

- In click-to-talk mode, the app publishes the audio stream of the participant and unsubscribes from the audio of subscribed streams.
- In mute mode, the app stops publishing the audio of the participant and subscribes to the audio for subscribed streams.

When the participant presses the **Click to talk** button, the `onMouseDown` handler calls the `startTalking()` method:

```
function startTalking() {
    document.getElementById("push-to-talk").onclick = stopTalking;
    document.getElementById("push-to-talk").value = "Click to mute";
    for (i = 0; i < session.subscribers.length; i++) {
        session.subscribers[i].subscribeToAudio(false);
    }
    publisher.publishAudio(true);
}
```

This function loops through the array of audio-video streams the page has subscribed to. This array is stored in the `subscribers` properties of the `session` object. Each of the elements in the array is a Subscriber object (defined by the OpenTok JavaScript library). The `subscribeToAudio(false)` method of the Subscriber object takes a boolean parameter as to whether to subscribe to the audio for the subscriber stream.

The `startTalking()` method also publishes the audio of the particpant by calling the `publishAudio(true)` method of the Publisher object that has been added to the page.

When the user clicks the **Click to mute** button, the `stopTalking()` method is invoked:

```
function stopTalking() {
    document.getElementById("push-to-talk").onclick = startTalking;
    document.getElementById("push-to-talk").value = "Click to talk";
    publisher.publishAudio(false);
    for (i = 0; i < session.subscribers.length; i++) {
        session.subscribers[i].subscribeToAudio(true);
    }
}
```

The `stopTalking()` method unpublishes the audio stream of the local participant and subscribes to the audio for subscribed streams. The method does this in a way similar to how the `startTalking()` method publishes the audio stream and unpublishes the audio of the subscribed streams.

It's important to make sure that while the local participant is in push-to-talk mode, all new subscribers who join the session are also muted. To accomplish this, the `subscribeToStream(stream)` method checks whether we are in push-to-talk mode:

```
var subscribeToAudio = true;
if(pushToTalk) {
    subscribeToAudio = false;
}

var subscriberProps = {
    width: SUBSCRIBER_WIDTH,
    height: SUBSCRIBER_HEIGHT,
    subscribeToAudio: subscribeToAudio
};
session.subscribe(stream, stubSpan.id, subscriberProps);
```

The participant can set the volume and mute all streams displayed on the page. Position the mouse over the lower-right corner of a displayed stream to see audio controls for that subscribed stream.

Improving echo suppression

When echo suppression is enabled, the OpenTok cloud continuously analyzes the conversation to decide who is leading the discussion at each moment in time, automatically suppressing audio input from other sources. Echo and progressive feedback artifacts are minimized.

Echo suppression works well:

- For small groups (generally up to 4 people)
- In discussion environments that encourage turn-taking

However, echo suppression can be fooled by:

- Background noise
- Unpredicted network latency
- Unusual audio response in microphone and speaker hardware

Push to talk

Using push to talk is like using a walkie talkie. Holding down a button while you talk ensures that a clear audio stream is transmitted.

Push to talk delivers high-quality audio by unsubscribing from the subscribed streams while the local participant is talking. This guarantees that the local participant's audio can be captured without audio artifacts. When the local participant is finished talking, releasing the button allows her to listen to the response.

Push to talk is effective, but it can be unnatural to use. Because the local participant cannot hear what people are saying when she is talking, it is easy to miss interjections and other cues that someone wants to respond. However, in some environment it is an effective approach for delivering high audio quality.

Wear headphones

The most effective technique to improving audio quality is to have all call participants wear headphones. Using headphones isolates the speaker output from the microphone input, removing the feedback loop that is otherwise present.

Unfortunately, not everyone is comfortable wearing headphones. If a session participant isn't wearing headphones, echo artifacts can be introduced into the session — and you may hear the echo, not the session participants.

If a participant chooses not to use heaphones and does not balance microphone and speaker levels carefully, it can degrade the audio in a multiparty session. If you have more than a few participants, audio quality can suffer significantly.

In these situations, a hybrid mode (some people wearing headphones, and others using push to talk) can be a very effective solution.

JavaScript Tutorial — Managing cameras and microphones

This tutorial shows how to use the TokBox API to select the camera and microphone used in an OpenTok session.

Testing the tutorial online

To run the tutorial:

1. Make sure that you have a webcam connected to your computer and configured to run.
2. Click the **Join session** link.

   This connects you to the OpenTok session and publishes an audio-video stream from your computer.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button. This grants tokbox.com access to your camera and microphone.

   You are now publishing a video stream to the session, and the video stream is displayed on the page.
3. Click the **Choose camera and microphone** link.

   The OpenTok Device Panel is displayed. This lets you select the camera and microphone used by the published audio-video stream.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button and then click the Close button. This grants TokBox.com access to your camera and microphone.
4. Click the **Change** button, under the **Cameras** section of the Device Panel.

   The Device Panel displays the video from each of your attached cameras.
5. Click the camera you want to use.

   The published stream now uses the new camera.
6. You can also select a microphone in the Device Panel.

Testing the code on your own computer

See the instructions in the Hello World tutorial.

Understanding the code

Note: You may want to look at the Basic Tutorial sample, if you have not already. It provides details on many of the basic concepts, like connecting to an OpenTok session, publishing an audio-video stream, and subscribing to streams.

To understand how the OpenTok JavaScript library is used, view the source for this browser page. You can also download the source for this Tutorial application (and for the other OpenTok tutorial applications) by clicking the **Download Source** link at the top of this page.

Using the Tokbox Device Panel

When you click the **Choose camera and microphone** link, the `onclick` handler calls the `toggleDevicePanel()` method. The `toggleDevicePanel()` method hides the Device Panel if it is already displayed:

```
if (panelShowing) {
    if (devicePanel) {
        deviceManager.removePanel(devicePanel);
    }
    document.getElementById("toggleDevicePanelLink").innerHTML =
                                    'Choose camera and microphone...';
    panelShowing = false;
}
```

Otherwise, if the Device Panel is not displayed, the `toggleDevicePanel()` method calls the `displayPanel()` method of a DeviceManager object to display the OpenTok Device Panel:

```
var newdiv = document.createElement("div");
newdiv.id = "devicePanel";
document.getElementById("devicePanelInset").appendChild(newdiv);
if (deviceManager == null) {
    deviceManager = TB.initDeviceManager(apiKey);
}
if (publisher) {
    devicePanel = deviceManager.displayPanel("devicePanel", publisher);
} else {
    devicePanel = deviceManager.displayPanel("devicePanel");
}
document.getElementById("toggleDevicePanelLink").innerHTML = 'Hide Device Panel';
panelShowing = true;
```

The `TB.initDeviceManager()` method returns a DeviceManager object. The TB object and the DeviceManager object are part of the TokBox API. The `displayPanel()` method of the DeviceManager object adds a TokBox Device Panel to an HTML `div` element. The `div` element is passed as the first parameter of the `displayPanel()` method. The second parameter references a Publisher object that should be updated based on camera and microphone selections in the device panel.

When you publish an audio-video stream, the `startPublishing()` method adds an event listener for the `settingsButtonClick` event. The Publisher object dispatches this event when the user clicks the settings button in the published stream display. (Mouse over the lower-right corner of the display to see the settings button.)

```
publisher.addEventListener("settingsButtonClick", showDevicePanel);
```

The `showDevicePanel()` calls the `toggleDevicePanel()` method, as needed, to display the Device Panel.

```
function showDevicePanel() {
    if (!panelShowing) {
        toggleDevicePanel()
    }
}
```

Detecting if a camera is connected

The TokBox API includes a `DeviceManager.detectDevices()` method that lets you detect the cameras and microphones attached to a computer.

This sample hides the `Join session` link if the user does not have a webcam connected. The `sessionConnectedHandler()` calls the `detectDevices()` method of a DeviceManager object. (This object is defined in the OpenTok JavaScript library.):

```
var deviceManager = TB.initDeviceManager();
deviceManager.addEventListener("devicesDetected", devicesDetectedHandler);
deviceManager.detectDevices();
```

When the code detects the connected devices, the DeviceManager object dispatches a `devicesDetected` event. This event invokes the a `devicesDetectedHandler()` function. This method shows the **Join session** link only if there are cameras and microphones connected:

```
function devicesDetectedHandler(event) {
    if (event.cameras.length == 0 || event.microphones.length == 0) {
        document.getElementById('joinSessionDiv').style.display = 'none';
        alert('No webcam or microphone is connected. You cannot participate in the
session.');
    } else {
        document.getElementById("joinSessionDiv").style.display = 'inline';
    }
}
```

Positioning the Device Panel in the page

This sample positions the Device Panel and its container `div` using styles defined in the samples.css file.

The default behavior of the `settingsButtonClick` event is to display the Device Panel in the center of the HTML page. However, this sample application cancels the default event by calling the `preventDefault()` method of the event object. The `settingsButtonClick` event handler then calls the `toggleDevicePanel()` function. This function adds the Device Manager to the page by calling the `displayPanel()` method of a DeviceManager object.

JavaScript Tutorial — Moderation and Signaling

This tutorial shows how to use the OpenTok JavaScript library to do the following:

- Moderate a session, by disconnecting a connection
- Have a connection send a signal to other connections

Testing the tutorial online

To run the tutorial:

1. Make sure that you have a webcam connected to your computer and configured to run.
2. Click the **Connect** link.

   This connects you to the OpenTok session. The TokBox API logs a message on the page: "Connected to the session."

3. Click the **Publish** link.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button and then click the Close button. This grants TokBox.com access to your camera and microphone.

   You are now publishing a video stream to the session, and the video stream is displayed on the page.

4. Copy the URL for this page into the Clipboard.
5. Open a new browser window, and have it open to the copied URL.
6. Click the **Connect** link (in the new browser window).

   The new page now connects to the session. Upon connection the video stream from the other page is displayed on the page.

7. Click the **Signal** link.

   The connection on the page sends a signal to all connections. Upon receiving the signal, each of the two pages displays a JavaScript alert ("Received a signal..."). Click the **OK** button in the alert dialog box.

8. Click the **Disconnect** link, underneath the displayed video stream (in the second browser window).

Testing the code on your own computer

See the instructions in the Hello World tutorial.

Understanding the code

Note: You may want to look at the Basic Tutorial sample, if you have not already. It provides details on many of the basic concepts, like connecting to an OpenTok session, publishing an audio-video stream, and subscribing to streams.

To understand how the OpenTok JavaScript library is used, View the source for this browser page. You can also download the source for this Tutorial application (and for the other OpenTok tutorial applications) by clicking the **Download Source** button at the top of this page.

Sending a signal to the TokBox session

When you click the **Signal** link in the page, the `a` tag calls the `signal()` function:

```
<a href="javascript:signal();" id="signalLink">Signal</a>
```

The `signal()` function calls the `signal()` method of the OpenTok Session object:

```
function signal() {
    session.signal();
}
```

The call to `session.signal()` sends a signal to the OpenTok session. Other connections to the session can set up event listeners to receive this signal.

Note: For information on code that initiates a TokBox session, connects to the session, see the Basic tutorial.

Receiving a signal from the TokBox session

When the page has connected to the TokBox session, (when the `session` object dispatches a `sessionConnected` event), the `sessionConnectedHandler()` function is called. This function includes code for adding an event listener for the `signalReceived` event:

```
session.addEventListener("signalReceived", signalReceivedHandler);
```

When any connection in the session sends a signal (using the `signal()` method of the Session object), the `session` object on this page dispatches a `signalReceived` event. The previous code registers the `signalReceivedHandler()` function.

The code defines a `signalReceivedHandler()` function as follows:

```
function signalReceivedHandler (event) {
    alert("Received a signal from connection " + event.fromConnection.connectionId);
}
```

Disconnecting a session

Upon connecting to a session, the code calls the `()` handler for the `sessionConnected` event:

```
function sessionConnectedHandler(event) {
    document.getElementById("publisherDiv").innerHTML += "<br/>Connected to the
session";
```

```
    for (var i = 0; i < event.streams.length; i++) {
        addStream(event.streams[i]);
    }
}
```

This function calls the `addStream()` function. This function constructs a `div` tag that contains the video for a stream as well as as **Force Disconnect** link for the stream:

```
function addStream(stream) {
    if (stream.connection.connectionId == session.connection.connectionId) {
        show("unpublishLink");
        return;
    }
    // Create the container for the subscriber
    var container = document.createElement('div');
    container.className = "subscriberContainer";
    var containerId = "container_" + stream.streamId;
    container.setAttribute("id", containerId);
    document.getElementById("subscriberBar").appendChild(container);

    // Create the div that will be replaced by the subscriber
    var div = document.createElement('div');
    var divId = stream.streamId;
    div.setAttribute('id', divId);
    div.style.float = "top";
    container.appendChild(div);

    // Create a div for the force disconnect link
    var forceDisconnect = document.createElement('div');
    forceDisconnect.style.float = "bottom";
    forceDisconnect.innerHTML =
          '<a href="#" onClick="javascript:forceDisconnectStream(\'' + stream.streamId
+ '\')">Force Disconnect</a>'
                + '<a href="#" onclick="javascript:forceUnpublishStream(\'' +
stream.streamId + '\')">Force Unpublish</a>'
    container.appendChild(forceDisconnect);

    subscribers[stream.streamId] = session.subscribe(stream, divId);
}
```

When the user clicks the **Force Disconnect** link, the `a` tag calls the `forceDisconnectStream()` function. This method calls the `forceDisconnect()` method of the Session object:

```
function forceDisconnectStream (streamId) {
    session.forceDisconnect(subscribers[streamId].stream.connection.connectionId);
}
```

The `forceDisconnect()` method of the Session object is part of the OpenTok JavaScript library. It tells the OpenTok session to disconnect the specified connection. In this case, the code maintains an array of streams based on stream IDs. (When generating the HTML containing the **Force Disconnect** link, the `addStream()` function added the stream ID as an identifier for a stream.

Forcing a participant to stop publishing a stream

The code described in the previous section adds a "Force Disconnect" link for each subscribed stream:

When the user clicks the **Force Unpublish** button, the `a` tag calls the `forceUnpublishStream()`

function. This method calls the `forceDisconnect()` method of the Session object:

```
function forceUnpublishStream(streamId) {
    session.forceUnpublish(subscribers[streamId].stream);
}
```

The `forceUnpublish()` method of the Session tells the OpenTok session to force the user to stop publishing the specified stream.

Note that `forceDisconnect()` and `forceUnpublish` methods are only available to users that have logged in with a token that has the moderation role assigned to it. You assign roles to tokens using the OpenTok server-side library. While testing your app, using the OpenTok staging server, you can use the `"moderator_token"` test token string, as is used in this demo.

Detecting when a user has been forced to disconnect

The `SessionDisconnectEvent` object has a `reason` property. This is set to `"forceDisconnected"` if the user's session was disconnected through a moderator calling `forceDisconnect()`. You can check this reason in the event handler:

```
if (event.reason == "forceDisconnected") {
    alert("A moderator has disconnected you from the session.");
}
```

Detecting when a user has been forced to unpublish

The `StreamEvent` object has a `reason` property. This is set to `"forceUnpublisheded"` if the stream session was disconnected through a moderator calling `forceUnpublish()`. You can check this reason in the event handler:

```
for (var i = 0; i < event.streams.length; i++) {
    removeStream(event.streams[i].streamId);
    if (event.streams[i].connection.connectionId == session.connection.connectionId &&
                      event.reason == "forceUnpublished") {
        alert("A moderator has stopped publication of your stream.");
        hide("unpublishLink");
        show("publishLink");
        publisher = null;
    }
}
```

JavaScript Tutorial — Targeting mobile platforms

This tutorial shows how to use the TokBox JavaScript library to view "auto-produced" streams. An auto-produced stream is one that multiplexes many all of the audio-video streams for a session into a single stream. This stream automatically switches the video to display the the active participant in the session (based on audio).

Subscribing to the multiplexed stream uses less bandwidth than subscribing to multiple individual streams in a session. This is an important consideration when running in a browser on a mobile device.

Testing the tutorial online

To run the tutorial, you will need to have at least two participants. Each participant will need to be on a networked computer that has a webcam and microphone. Once you have everyone ready, do the following:

1. Open the page in a browser on one machine.

   When you first open the page, it connects to an OpenTok session. Running in the browser on a desktop computer, the height and width of the content are intended to resemble a mobile phone in portrait orientation.

   If there are any participants publishing audio-video streams to the session, the page displays an auto-produced multiplexed stream. If no users are currently publishing audio-video stream, then no multiplexed stream is displayed.

2. Click the **Start publishing** link.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button and then click the Close button. This grants tokbox.com access to your camera and microphone.

   You are now publishing a video stream to the OpenTok session, and the video stream is displayed in the lower-lefthand corner of the app.

3. Repeat the previous steps on the other computers.

   As the OpenTok session detects audio activity on the different computers, it switches the video in the auto-produced stream.

4. Now, run the app in the web browser on an Android device that supports Flash Player 10.1.

   The app shows the auto-produced stream. You can rotate the device, and the app orientation rotates.

   Flash Player for Android does not allow access to the camera, so you cannot publish a video from the device.

For a list of supported Android devices, see the Adobe list of Flash platform certified devices.

Remember that the switching is voice-activated. Have the participants speak up in order to see the auto-produced multiplexing in action.

Uses for auto-production

Using the auto produced stream lets you bring a six-person video conference across a limited-bandwidth network connection, such as that in a browser running on a mobile device.

Auto-production lets you to funnel all the participants in a session through a limited number of video streams. The TokBox cloud acts as your "producer in the sky". Depending on how you set up the session, the producer will automatically switch between video streams based on who is talking. Or it can cycle through the streams on a regular basis — in the same way a security camera might.

The auto-producer may not focus on the correct individual when two participants talk at the same time. However, the multiplexed stream still includes all the audio from each participant.

Testing the code on your own computer

See the instructions in the Hello World tutorial.

Understanding the code

Note: You may want to look at the Basic Tutorial sample, if you have not already. It provides details on many of the basic concepts, like connecting to an OpenTok session, publishing an audio-video stream, and subscribing to streams.

To understand how the OpenTok JavaScript library is used, view the source for this browser page. You can also download the source for this Tutorial application (and for the other OpenTok tutorial applications) by clicking the **Download Source** link at the top of this page.

Setting up a session to use multiplexed streams

When you use the OpenTok server-side library to create a session, you can request the number of multiplexed streams to be auto produced. For example, using the OpenTOK PHP library, you can set the `multiplexer.numOutputStreams` property. This property is defined as the `MULTIPLEXER_NUMOUTPUTSTREAMS` constant in the SessionPropertyConstants.php file:

```
require_once 'SDK/API_Config.php';
require_once 'SDK/OpenTokSDK.php';
require_once 'SDK/SessionPropertyConstants.php';

$apiObj = new OpenTokSDK(API_Config::API_KEY, API_Config::API_SECRET);

$session = $apiObj->create_session($_SERVER["REMOTE_ADDR"],
array(SessionPropertyConstants::MULTIPLEXER_NUMOUTPUTSTREAMS=>2));

echo $session->getSessionId();
echo "
";
```

```
echo $apiObj->generate_token();
```

In this example, you request a session that has two multiplexed streams. The OpenTok server returns a Session XML file that includes the matching number of output streams (defined in the `numOutputStreams` element:

```
<Sessions>
  <Session>
    <session_id>1cbd44eb05e7d8e6e05c5c3aaa0ff295d744237f</session_id>
    <partner_id>1234</partner_id>
    <create_dt>2010-11-11 19:51:21+00:00</create_dt>
    <properties>
      <multiplexer>
        <numOutputStreams>1</numOutputStreams>
      </multiplexer>
    </properties>
  </Session>
</Sessions>
```

The session with the given session ID automatically has the given number of multiplexed, auto-produced streams (in this case, only one).

Subscribing to streams

When the page connects to the OpenTok session, the Session object dispatches a `sessionConnected` event. The `sessionConnected()` function is the event handler for this event. The `streams` property of the `sessionConnected` event is an array of Stream objects (defined in the OpenTok ActionScript library):

```
function sessionConnectedHandler(event) {
    subscribeToStreams(event.streams);

    if (session.capabilities.publish) {
        setStatus("You can start publishing.");
        show("publishLink");
    } else {
        setStatus("You are connected.");
    }
}
```

The `capabilities` property of the Session object defines the capabilites for the current user. The `capabilities.publish` property is set to `false` if a user is unable to publish. Users whose token has not been granted a publishing role cannot publish. Also, users running the app in a browser on Android cannot publish. (Flash Player for Android does not grant access to cameras on the Android device.) Users who cannot publish can still connect to OpenTok sessions and subscribe to streams.

The `subscribeToStreams()` function and the `subscribeToStreams()` function subscribe to streams.

Each Stream object has a `type` property. And if this type is set to `"multiplexed"`, the code sets the `subscribeToAudio` property of an object to `false`. The `subscribeToStream()` function will then use this object as the `subscriberProperties` parameter of the `subscribe()` method of the session object. Similarly, for each individual's stream (other than the one the user publishes), it subscribes to the stream without subscribing to video (in audio-only mode). It adds the individual (basic) videos to the `subscribers` DIV element, which is not visible.

```
function subscribeToStreams(streams) {
    for (var i = 0; i < streams.length; i++) {
        var stream = streams[i];

        var groupContainer = document.getElementById('group');
        if (stream.type == "multiplexed" && groupContainer.children.length == 0) {
            var groupWidth = window.innerWidth - 20;
            var groupHeight = window.innerHeight - 60;

            var subscriberProps = { width: groupWidth,
                                    height: groupHeight,
                                    subscribeToAudio: false };

            subscribeToStream(stream, groupContainer, subscriberProps);
        } else if (stream.connection.connectionId != session.connection.connectionId) {
            // Create an audio only subscriber
            var parentDiv = document.getElementById("subscribers");
            subscribeToStream(stream, parentDiv, {subscribeToVideo:false});
        } else {
            setStatus("You are publishing.");
            hide("publishLink");
            show("unpublishLink");
        }
    };
}
```

The `subscribeToStream` function calls the `subscribe()` method of the Session object. This subscribes to the stream, and adds its video to the specified container DIV element:

```
function subscribeToStream(stream, parentDiv, subscriberProps) {
    var subscriberDiv = document.createElement("div");
    subscriberDiv.id = "opentok_subscriber_" + stream.streamId;
    parentDiv.appendChild(subscriberDiv);

    session.subscribe(stream, subscriberDiv.id, subscriberProps);
}
```

The Session object dispatches a `streamCreated` event when new streams are created. The `streamCreatedHandler()` function is the event handler, and it also calls the `subscribeToStreams()` function to add new streams.

The `streamDestroyedHandler()` function unsubscribes from individual streams. It iterates through the `streams` array of the StreamEvent in much the same way that the `streamCreatedHandler()` method does.

JavaScript Tutorial — Resizing video

This tutorial shows how you can control the size of the OpenTok video display, just as you would any other HTML element. Also, it demonstrates that you can show an OpenTok video multiple times in the web page.

Testing the tutorial online

To run the tutorial:

1. Make sure that you have a webcam connected to your computer and configured to run.
2. Click the **Start publishing** link.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button. This grants TokBox.com access to your camera and microphone.

   You are now publishing a video stream to the session, and the video stream is displayed multiple times on the page, in different sizes.

As you follow the diagonal up and to the left from that corner, you can see scaled-down versions of the published stream, each of which maintains the original 4:3 aspect ratio.

For containers that display a stream with dimensions that do not match the 4:3 aspect ratio, the video image is be clipped. The video image is clipped so that the center-most part of the image is displayed. You can see this in some of the video displays of the movie.

Testing the code on your own computer

See the instructions in the Hello World tutorial.

Understanding the code

Note: You may want to look at the Basic Tutorial sample, if you have not already. It provides details on many of the basic concepts, like connecting to an OpenTok session, publishing an audio-video stream, and subscribing to streams.

A `subscriber_width` array and a `subscriber_height` array define the widths and heights to use for the displayed videos:

```
var subscriber_width = [120, 160, 220];
var subscriber_height = [90, 120, 165];
```

When you publish a stream, the Session object (defined in the Tokbox JavaScript library) dispatches a `streamCreated` event. The `streamCreatedHandler()` is the event handler for this event.

```
function streamCreatedHandler(event) {
    for (var i = 0; i < event.streams.length; i++) {
        if (event.streams[i].connection.connectionId ==
event.target.connection.connectionId) {
```

```
            for (y = 0; y < 3; y++) {
                for (x = 0; x < 3; x++) {
                    // Create a div for the subscriber to replace
                    var parentDiv = document.getElementById("cell_" + x + "_" + y);
                    var subscriberDiv = document.createElement("div");
                    subscriberDiv.id = "opentok_subscriber_" + x + "_" + y;
                    parentDiv.appendChild(subscriberDiv);

                    var subscriberProps = {width: subscriber_width[x],
                                           height: subscriber_height[y],
                                           audioEnabled: false};
                    event.target.subscribe(event.streams[i], subscriberDiv.id,
subscriberProps);
                }
            }
        }
    }
}
```

The StreamEvent object, which defines the `streamCreated` event, has a `streams` property. This property is an array of Stream objects. The `streamCreatedHandler()` function checks each Stream object in the array to see if it matches the stream you just published. It does this by comparing the connection ID for the stream with that of your own connection. (Each user's connection to the session has a connection ID.) For more information on Session and Stream objects, see the Basic tutorial.

If the code finds a stream that matches your session (one that you published), it iterates through the cells of the HTML table. For each cell, the code creates a container `div` and calls the `subscribe()` method of the Session object. This adds the video to the page.

The last parameter of the `subscribe()` method is the `properties` parameter. The `width` and `height` properties of this object determine the width and height of the video displayed. Note that the code obtains these values from the `subscriber_width` and `subscriber_height` arrays:

```
var subscriberProps = {width: subscriber_width[x],
                       height: subscriber_height[y],
                       audioEnabled: false};
```

JavaScript Tutorial — Audio-only and Video-only

This tutorial shows how to publish a video as audio-only or video-only. It also shows how to change the local playback of a subscribed stream to audio-only or video-only.

For more information on controlling audio, see the Controlling Audio tutorial.

Testing the tutorial online

To run the tutorial:

1. Make sure that you have a webcam connected to your computer and configured to run.
2. Click the **Connect** button.
3. Wait for the page to connect to the session. Then choose the **Audio-only** option and click the **Start Publishing** button.

   If the **Flash Player Settings** dialog is displayed, click the **Allow** button and then click the Close button. This grants TokBox.com access to your camera and microphone.

   You are now publishing an audio-only stream to the session, and the video stream placeholder is displayed on the page.
4. Copy the URL for this page into the Clipboard.
5. Open a new browser window, and have it open to the copied URL.

   Better yet ... have another user open the URL in a browser on another computer. If you are connecting on the same computer, be sure to wear headphones, to avoid audio feedback.

   The new page now connects to the session. Upon connection the stream, the video-only placeholder from the first page is displayed on the new page.
6. Go back to the first page and click the **Turn on my video** link. Then click the **Turn off my audio** link.
7. Go to the second page and note that the stream is now video-only.
8. Go back to the first page and click the **Turn on my audio** link.
9. Go to the second page and note that audio and video are published again.
10. Click the **Turn off audio** link for the subscribed video.

    The second page now retrieves a video-only stream.
11. Click the **Turn on audio** link for the subscribed video.

    The second page now plays both audio and video.
12. Click the **Turn off video** link for the subscribed video.

The second page now retrieves an audio-only stream.

Testing the code on your own computer

See the instructions in the Hello World tutorial.

Understanding the code

Note: You may want to look at the Basic Tutorial sample, if you have not already. It provides details on many of the basic concepts, like connecting to an OpenTok session, publishing an audio-video stream, and subscribing to streams.

To understand the OpenTok JavaScript library is used, View the source for this browser page. You can also download the source for this Tutorial application (and for the other OpenTok tutorial applications) by clicking the **Download Source** button at the top of this page.

Setting the initial publishing mode: audio-only, video-only, or audio/video

When the user clicks the **Start publishing** button, it invokes the click event handler, the `startPublishing()` function. It sets up the div objects and other objects needed to start publishing, which you do by calling the `publish()` method of the Session object. The `publish()` method has an optional `properties` parameter. The object you pass to this parameter has optional `publishAudio` and `publishVideo` properties, which determine whether the stream initially publishes audio or video (or both). The `startPublishing` function sets these properties based on the user's radio-button selections. It then passes the publish properties to the `publish()` method of the Session object:

```
var publisherProperties = new Object();
if (document.getElementById("pubAudioOnly").checked) {
        publisherProperties.publishVideo = false;
}
if (document.getElementById("pubVideoOnly").checked) {
        publisherProperties.publishAudio = false;
}

publisher = session.publish(publisherDiv.id, publisherProperties);
```

The method also calls the `getPublisherControls()` function, which creates a set of links to toggle video and audio for the published stream.

```
function getPublisherControls() {
    sessionControlsDiv = document.createElement('div');
    sessionControlsDiv.innerHTML =
        '' +
        '' +
        '' +
        ''
    return sessionControlsDiv;
}
```

Toggling publishing of audio and video

When the stream you publish is created (when the Session dispatches the `streamCreated` event for your stream), the code makes the **Turn on/off my audio** and **Turn on/off my video** links visible. The event listener for the `streamCreated` event determines which links (for example "Turn on my audio" or "Turn off my audio") to display, based on the user's settings.

The `onClick` handler for the **Turn off my audio** link calls the following code:

```
publisher.publishAudio(false)
```

The `onClick` handler for the **Turn on my audio** link calls the following code:

```
publisher.publishAudio(true)
```

The `publishAudio()` method of the Publisher object turns publishing of audio on and off based on the value you pass (`true` or `false`).

Similarly, the `onClick` handler for the **Turn off my video** link calls `publisher.publishVideo(false)`. And the `onClick` handler for the **Turn on my video** link calls `publisher.publishVideo(true)`.

Toggling audio and video in subscribed streams

When another user's stream is created (when the Session dispatches the `streamCreated` event for another user's stream), the code makes the **Turn off audio** and **Turn off video** links visible for the subscribed stream. The visibility of these links is based on the hasAudio property of the Stream object. (If the stream does not have audio, the links are hidden.)

```
var audioControlsDisplay;
if (stream.hasAudio) {
        audioControlsDisplay = "block";
} else {
        audioControlsDisplay = "none";
}
var videoControlsDisplay;
if (stream.hasVideo) {
        videoControlsDisplay = "block";
} else {
        videoControlsDisplay = "none";
}
actionDiv.innerHTML =
        '<span id="' + streamId +'-audioControls" style="display:' +
audioControlsDisplay + '"> \
        <a href="#" id="'+streamId+'-audioOff"
onclick="turnOffHerAudio(\''+streamId+'\');" style="display:block">Turn off audio</a>\
   <a href="#" id="'+streamId+'-audioOn" onclick="turnOnHerAudio(\''+streamId+'\')"
style="display:none">Turn on audio</a>\
   </span> \
        <span id="' + streamId +'-videoControls" style="display:' +
videoControlsDisplay + '"> \
   <a href="#" id="'+streamId+'-videoOff" onclick="turnOffHerVideo(\''+streamId+'\')"
style="display:block">Turn off video</a>\
   <a href="#" id="'+streamId+'-videoOn" onclick="turnOnHerVideo(\''+streamId+'\')"
style="display:none">Turn on video</a>\
   </span>';
```

The event handler for the `streamPropertyChanged` event toggles the visibility of the subscriber audio and video links, based on the changed property and the new value:

```
function streamPropertyChangedHandler(event)
{
        var stream = event.stream;
        if (event.changedProperty == "hasAudio") {
                var audioControls = document.getElementById(stream.streamId + "-
audioControls");
                if (event.newValue == true) {
                        audioControls.style.display = "block";
                } else {
                        audioControls.style.display = "none";
                }
        }
        if (event.changedProperty == "hasVideo") {
                var audioControls = document.getElementById(stream.streamId + "-
videoControls");
                if (event.newValue == true) {
                        audioControls.style.display = "block";
                } else {
                        audioControls.style.display = "none";
                }
        }
}
```

The `onClick` handler for the **Turn off audio** link calls the following code:

```
subscriber.subscribeToAudio(false)
```

The `subscribeToAudio()` method of the Subscriber object turns subscribing to audio on and off based on the value you pass (`true` or `false`). Note that this method only affects inward streaming of audio on the local page. It does not affect the publisher's streaming or other user's pages.

The `onClick` handler for the **Turn on audio** link calls the following code:

```
subscriber.subscribeToAudio(true)
```

Similarly, the `onClick` handler for the **Turn off video** link calls `subscriber.subscribeToVideo(false)`. And the `onClick` handler for the **Turn on video** link calls `subscriber.subscribeToVideo(true)`.