

From the computer screen (screen sharing) in a browser via WebRTC

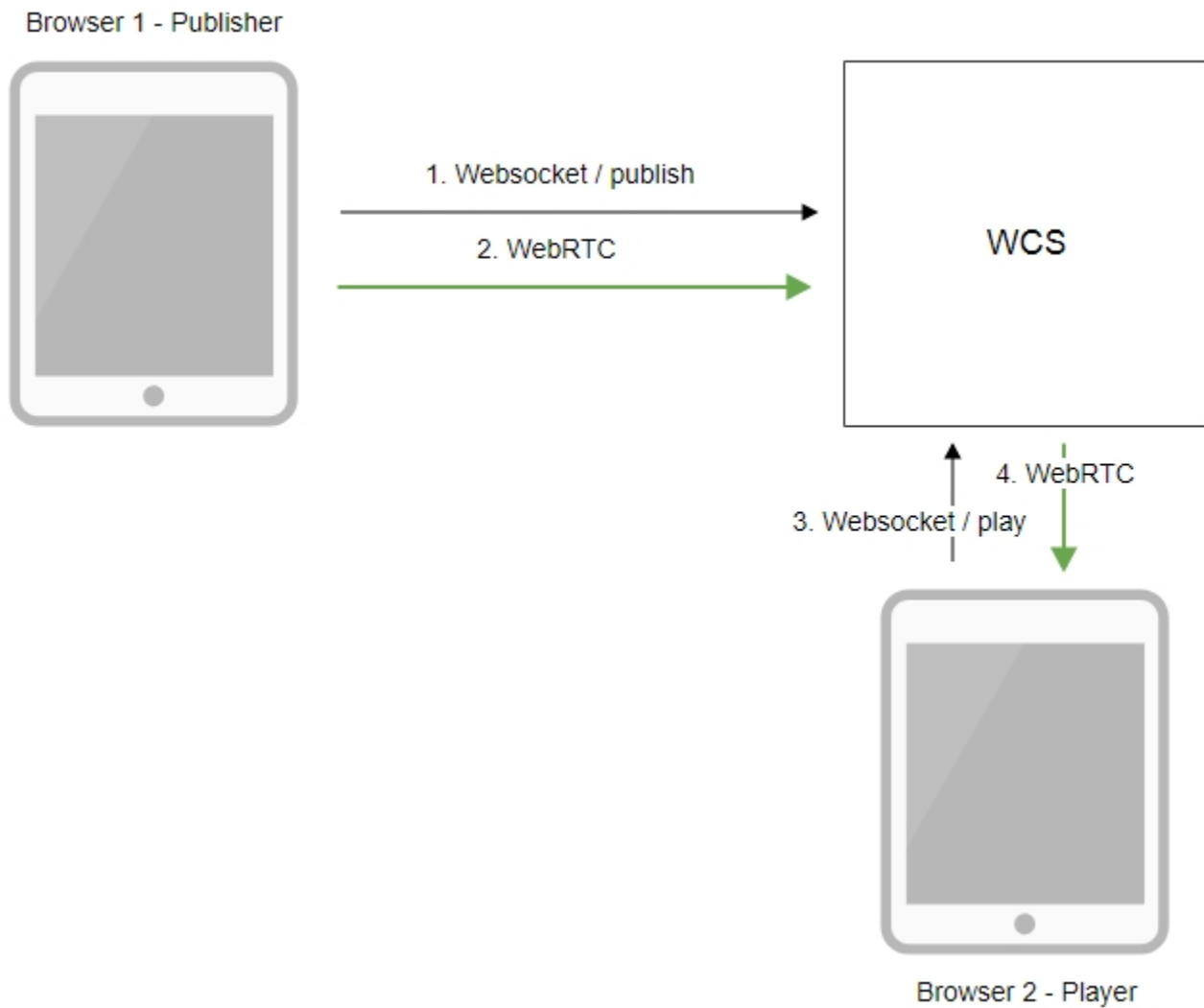
- [Overview](#)
 - [Supported platforms and browsers](#)
 - [Operation flowchart](#)
- [Quick manual on testing](#)
 - [Capturing a video stream from the computer screen and preparing for publishing](#)
- [Call flow](#)
- [To developer](#)
 - [Extension for Google Chrome with publication in Chrome Store](#)
 - [Create a Google account](#)
 - [Register as Chrome Web Store Developer](#)
 - [Customization for your domain](#)
 - [Package the extension](#)
 - [Publish the extension](#)
 - [Extension Inline Installation](#)
 - [Configuring the client](#)
 - [Media source parameters](#)
 - [Parameter list](#)
 - [System sound capture in Chrome browser](#)
 - [Capture source \(screen or window\) management in Firefox browser](#)
 - [Screen sharing without extension](#)
 - [Firefox browser](#)
 - [Chromium based browsers](#)
 - [Safari browser in MacOS](#)
 - [Known limits](#)
- [Known issues](#)

Overview

Supported platforms and browsers

	Chrome	Firefox
Windows	+	+
Mac OS	+	+

Operation flowchart



1. The browser establishes a connection to the server via the Websocket protocol and sends the publish command.
2. The browser captures the screen and sends a WebRTC stream to the server.
3. The second browser establishes a connection also via Websokcet and sends the play command.
4. The second browser receives the WebRTC stream and plays the stream on the page.

Quick manual on testing

Capturing a video stream from the computer screen and preparing for publishing

1. For the test we use the demo server at demo.flashphoner.com and the Screen Sharing web application in the Chrome browser
<https://demo.flashphoner.com/client2/examples/demo/streaming/screen-sharing/screen-sharing.html>

Screen Sharing

Chrome extension

Install Now

FPS

Width

Height

30

640

480

W/o extension?

Use Mic?

Mic

☒

☐

Default - Microphone

My Screen

wss://demo.flashphoner.com:8443/3a437ca9

Start

Preview

2. Click the "Start" button. The browser asks for permission to access the screen, and screen capturing starts, then the stream is publishing:

Screen Sharing

Chrome extension

Install Now

FPS

Width

Height

30

640

480

W/o extension?

Use Mic?

Mic

☒

☐

Default - Microphone

My Screen

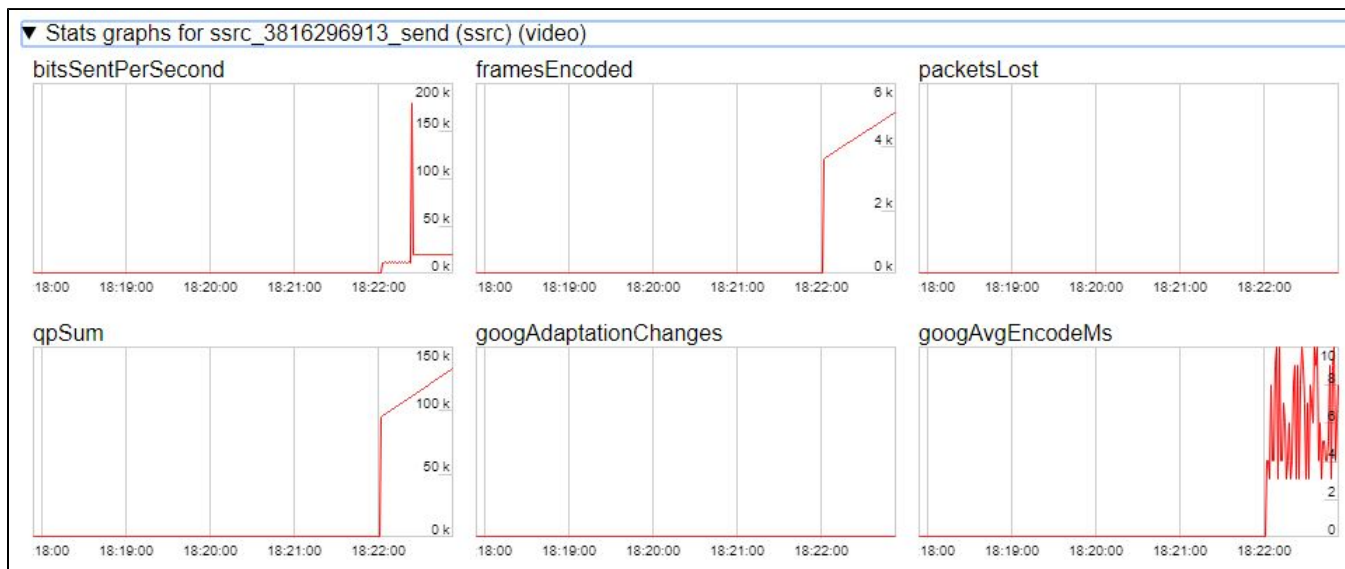
wss://demo.flashphoner.com:8443/3a437ca9

Stop

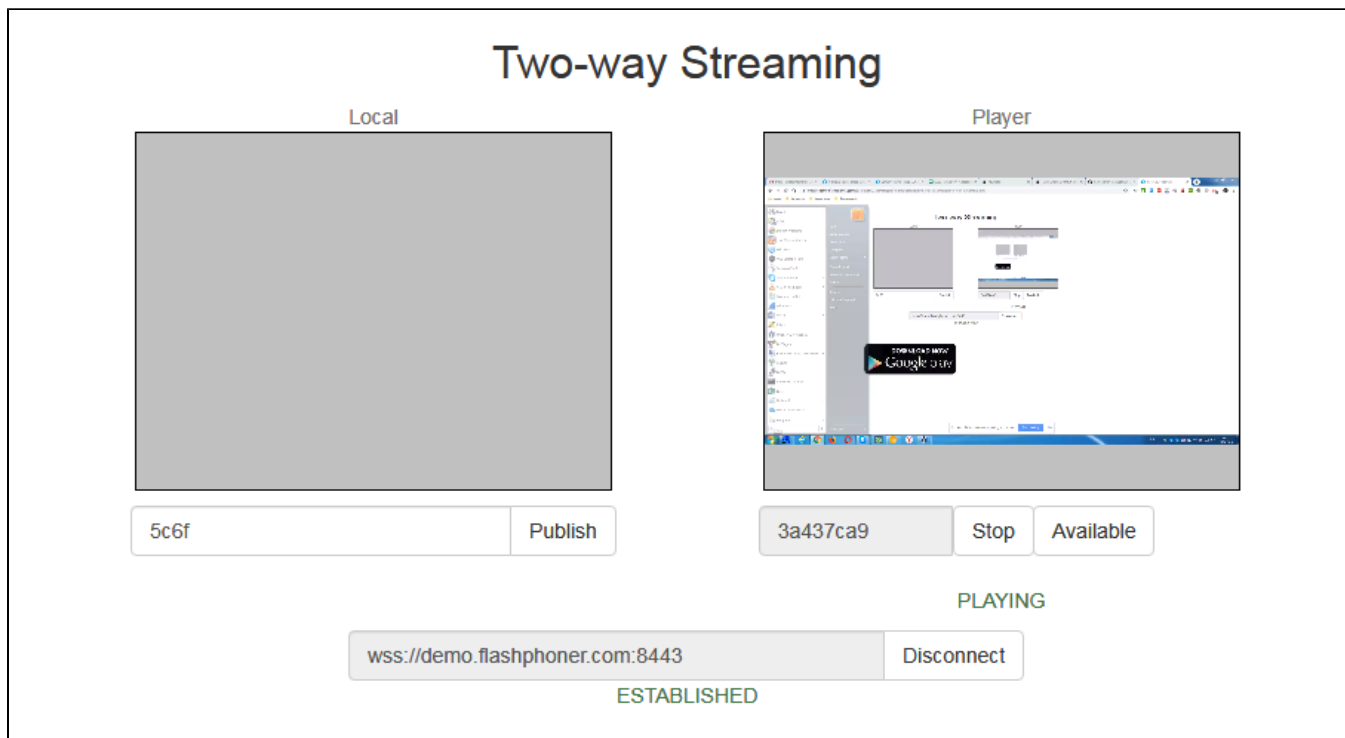
PUBLISHING

Preview

3. Make sure the stream is sent to the server and the system operates normally in <chrome://webrtc-internals>



4. Open Two Way Streaming in a new window, click Connect and specify the stream id, then click Play.



5. Playback diagrams in <chrome://webrtc-internals>

▼ Stats graphs for ssrc_50170578_recv (ssrc) (video)

cname:rtp/video/734701b0-4ec4-11e8-bfde-69f9687529f1

bitsReceivedPerSecond



framesDecoded



packetsLost



qpSum



googCaptureStartNtpTimeMs



googCurrentDelayMs

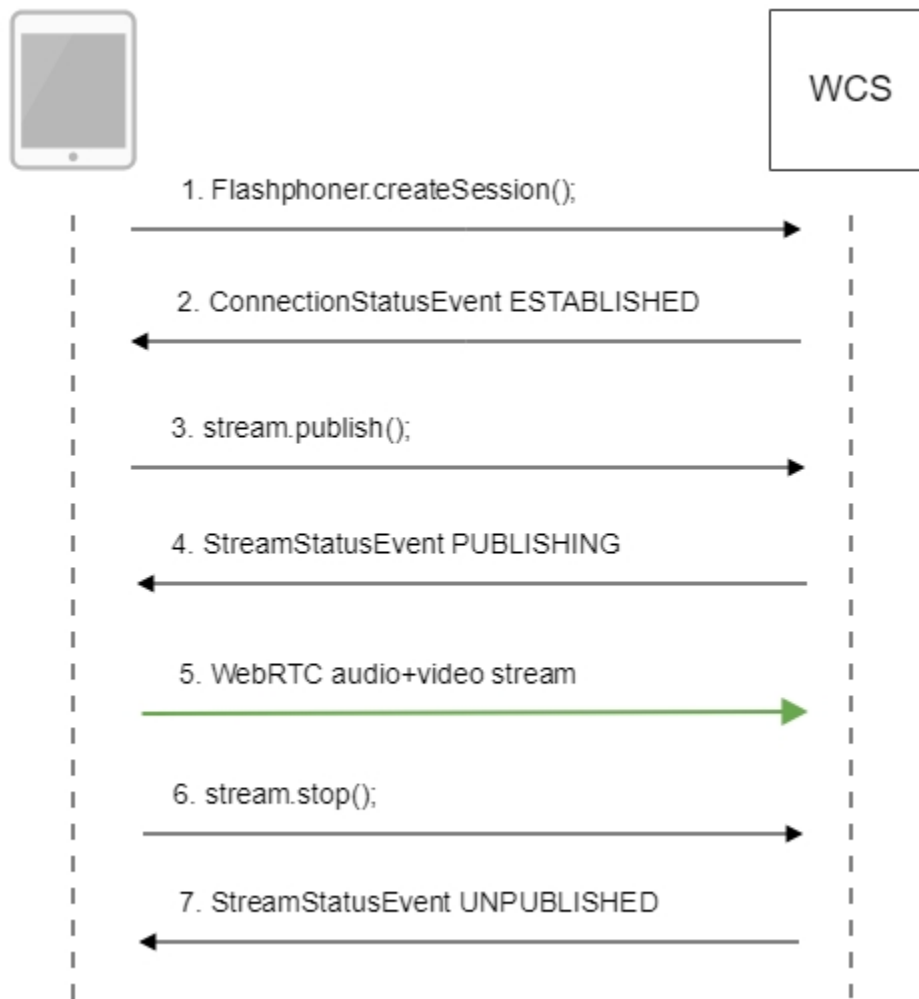


Call flow

Below is the call flow when using the Screen Sharing example

[screen-sharing.html](#)

[screen-sharing.js](#)



1. Checking if the extension install is required

Browser.isFirefox(), Browser.isChrome();[code](#)

```

if (Browser.isFirefox()) {
    $("#installExtensionButton").show();
    ...
} else if (Browser.isChrome()) {
    $('#mediaSourceForm').hide();
    interval = setInterval(function() {
        chrome.runtime.sendMessage(extensionId, {type: "isInstalled"}, function (response) {
            if (response) {
                $("#extension").hide();
                clearInterval(interval);
                onExtensionAvailable();
            } else {
                (inIframe()) ? $("#installFromMarket").show() : $("#installExtensionButton").show();
            }
        });
    }, 500);
} else {
    $("#notify").modal('show');
    return false;
}

```

2. Establishing a connection to the server.

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

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function(session){
    //session connected, start streaming
    startStreaming(session);
}).on(SESSION_STATUS.DISCONNECTED, function(){
    setStatus(SESSION_STATUS.DISCONNECTED);
    onStoped();
}).on(SESSION_STATUS.FAILED, function(){
    setStatus(SESSION_STATUS.FAILED);
    onStoped();
});
```

3. Receiving from the server an event confirming successful connection.

ConnectionStatusEvent ESTABLISHED[code](#)

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function(session){
    //session connected, start streaming
    startStreaming(session);
    ...
});
```

4. Publishing the stream.

stream.publish();[code](#)

```
session.createStream({
    name: streamName,
    display: localVideo,
    constraints: constraints
}).on(STREAM_STATUS.PUBLISHING, function(publishStream){
    ...
}).on(STREAM_STATUS.UNPUBLISHED, function(){
    ...
}).on(STREAM_STATUS.FAILED, function(stream){
    ...
}).publish();
```

5. Receiving from the server an event confirming successful publishing.

StreamStatusEvent, статус PUBLISHING[code](#)

```

session.createStream({
  name: streamName,
  display: localVideo,
  constraints: constraints
}).on(STREAM_STATUS.PUBLISHING, function(publishStream){
  ...
  setStatus(STREAM_STATUS.PUBLISHING);
  //play preview
  session.createStream({
    name: streamName,
    display: remoteVideo
  }).on(STREAM_STATUS.PLAYING, function(previewStream){
    document.getElementById(previewStream.id()).addEventListener('resize', function(event){
      resizeVideo(event.target);
    });
    //enable stop button
    onStart(publishStream, previewStream);
  }).on(STREAM_STATUS.STOPPED, function(){
    publishStream.stop();
  }).on(STREAM_STATUS.FAILED, function(stream){
    //preview failed, stop publishStream
    if (publishStream.status() == STREAM_STATUS.PUBLISHING) {
      setStatus(STREAM_STATUS.FAILED, stream);
      publishStream.stop();
    }
  }).play();
}).on(STREAM_STATUS.UNPUBLISHED, function(){
  ...
}).on(STREAM_STATUS.FAILED, function(stream){
  ...
}).publish();

```

6. Sending the audio-video stream via WebRTC

7. Stopping publishing the stream.

`stream.stop();`[code](#)


```

session.createStream({
  name: streamName,
  display: localVideo,
  constraints: constraints
}).on(STREAM_STATUS.PUBLISHING, function(publishStream){
  /*
   * User can stop sharing screen capture using Chrome "stop" button.
   * Catch onended video track event and stop publishing.
   */
  document.getElementById(publishStream.id()).srcObject.getVideoTracks()[0].onended = function (e) {
    publishStream.stop();
  };
  ...
  setStatus(STREAM_STATUS.PUBLISHING);
  //play preview
  session.createStream({
    name: streamName,
    display: remoteVideo
  }).on(STREAM_STATUS.PLAYING, function(previewStream){
    ...
  }).on(STREAM_STATUS.STOPPED, function(){
    publishStream.stop();
  }).on(STREAM_STATUS.FAILED, function(stream){
    //preview failed, stop publishStream
    if (publishStream.status() == STREAM_STATUS.PUBLISHING) {
      setStatus(STREAM_STATUS.FAILED, stream);
      publishStream.stop();
    }
  }).play();
  ...
}).publish();

```

8. Receiving from the server an event confirming unpublishing of the stream.

StreamStatusEvent, `cratyc UNPUBLISHED`[code](#)

```

session.createStream({
  name: streamName,
  display: localVideo,
  constraints: constraints
}).on(STREAM_STATUS.PUBLISHING, function(publishStream){
  ...
}).on(STREAM_STATUS.UNPUBLISHED, function(){
  setStatus(STREAM_STATUS.UNPUBLISHED);
  //enable start button
  onStopped();
}).on(STREAM_STATUS.FAILED, function(stream){
  ...
}).publish();

```

To developer

The screen sharing function can be used to publish a video stream that demonstrates the desktop or an application window. WCS API uses a Chrome extensions for screen sharing. Firefox browser since version 52 does not require an extension.

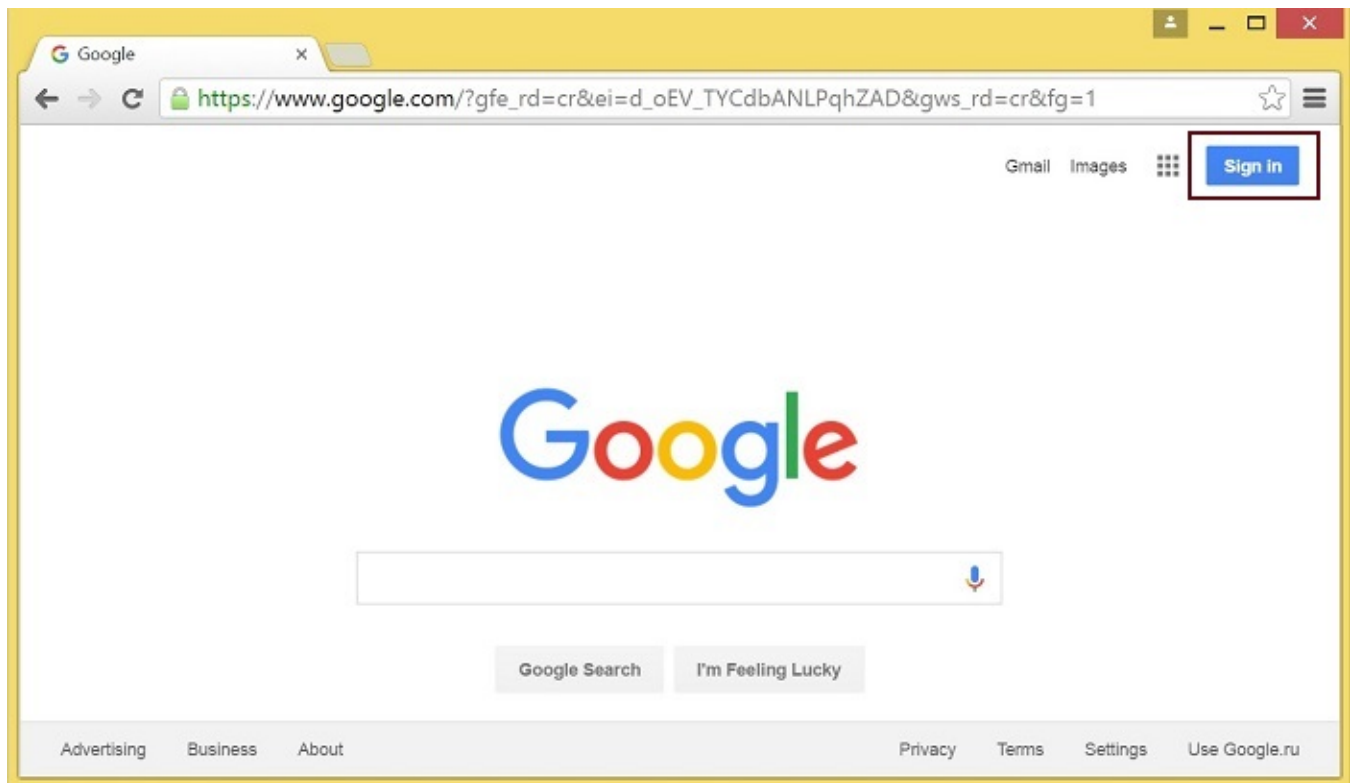
Extension for Google Chrome with publication in Chrome Store

Source code for the extension is available by the following link:

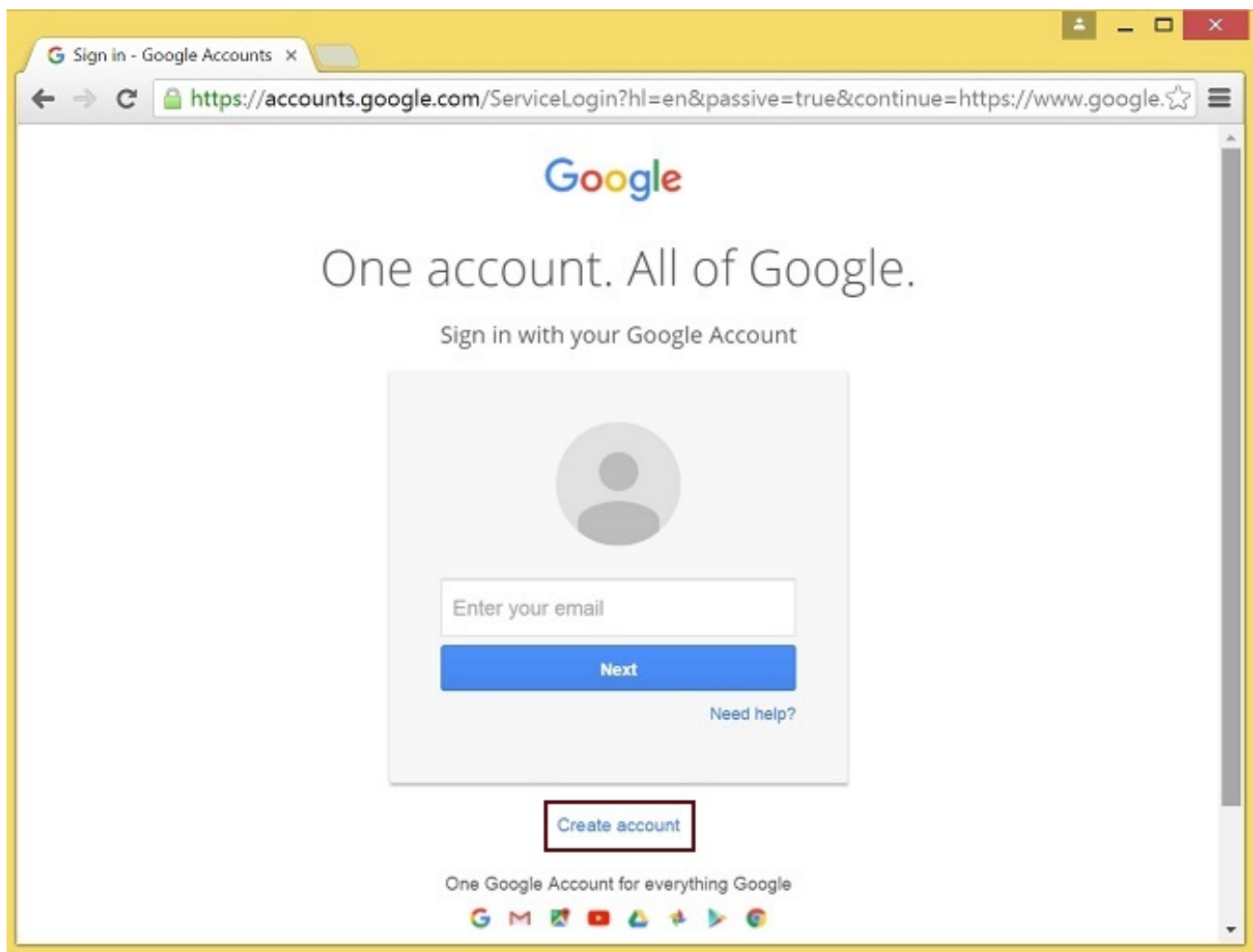
[Chrome Screen Sharing Extension](#)

Create a Google account

1. Go to google.com and click 'Sign in' button



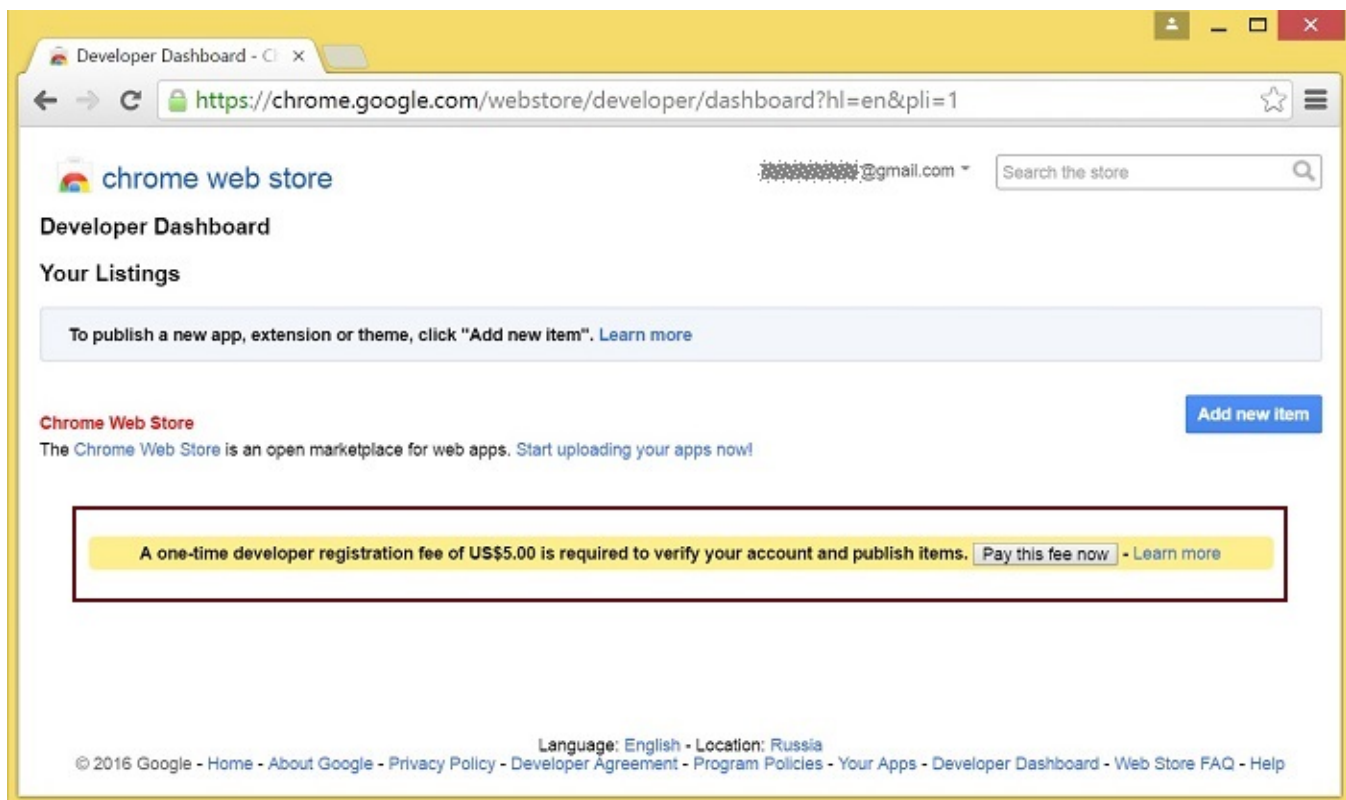
2. On the 'Sign in' page, click 'Create account' link



3. 'Create your Google Account' page will be opened
Fill the required fields and click 'Next step' button to create the account.

Register as Chrome Web Store Developer

1. Sign in to [Chrome Developer Dashboard](#) with the created Google account
2. Pay a one-time \$5 developer signup fee



Customization for your domain

Follow the procedures described below to use the extensions with your domain.
Edit manifest file manifest.json of the Chrome extension.

Change:

- name
- author
- description
- homepage_url
- under "externally_connectable": "matches" change flashphoner.com to your domain

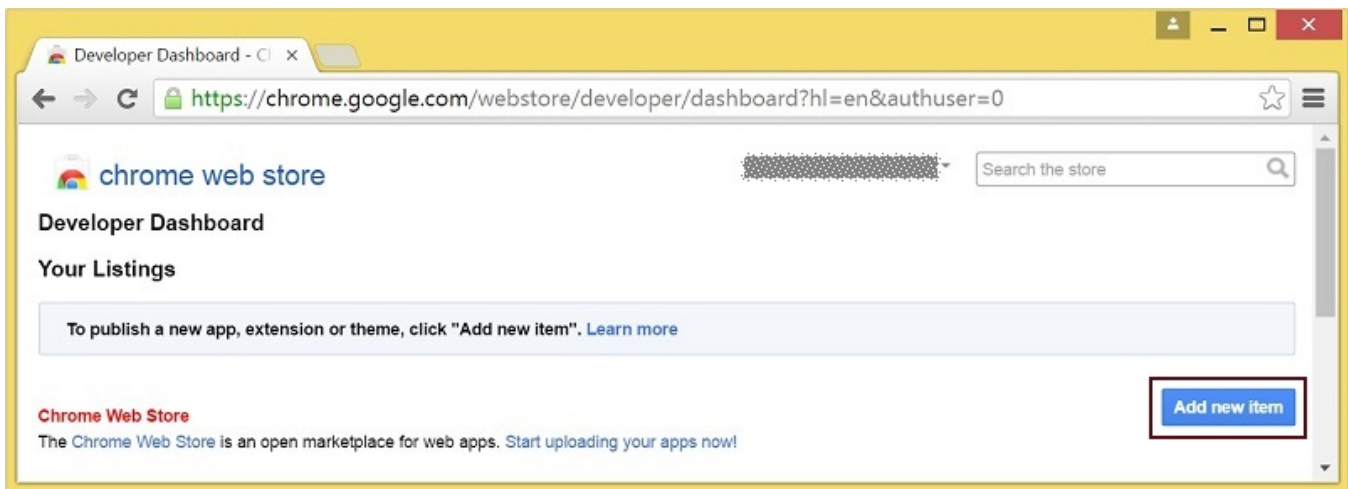
Save your icons for the extension to chrome-extension directory and edit the file names in "icons" and "web_accessible_resources". (For more information, see [Manifest - Icons](#) and [Supplying Images](#).)

Package the extension

Package files from chrome-extension folder into ZIP archive.

Publish the extension

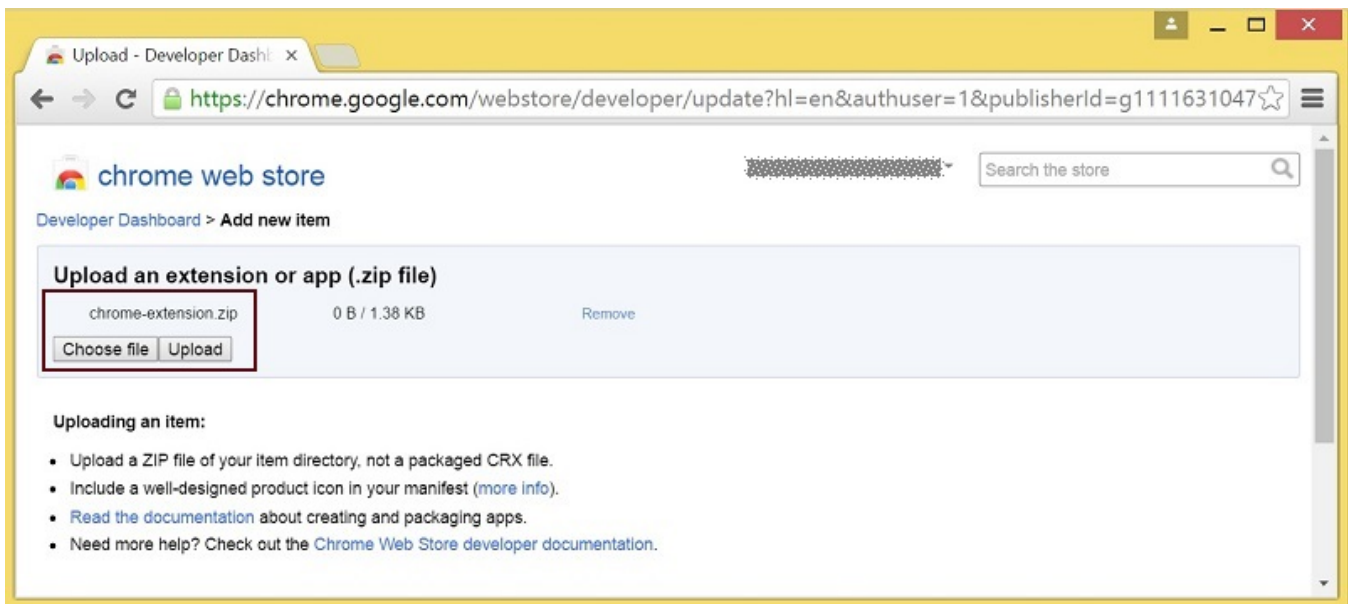
1. Sign in to [Chrome Developer Dashboard](#)
2. In the dashboard, click 'Add new item' button



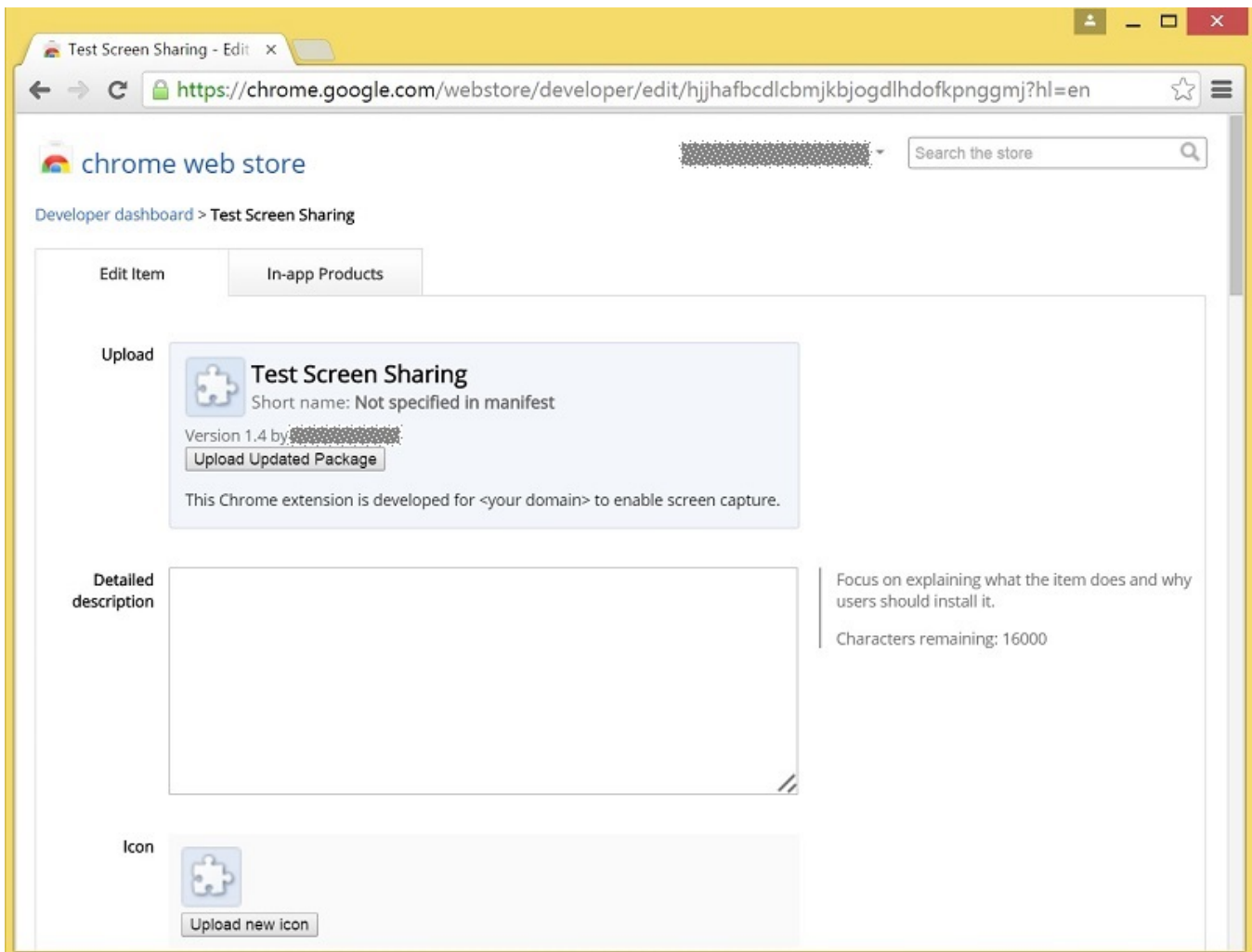
3. Accept the developer agreement

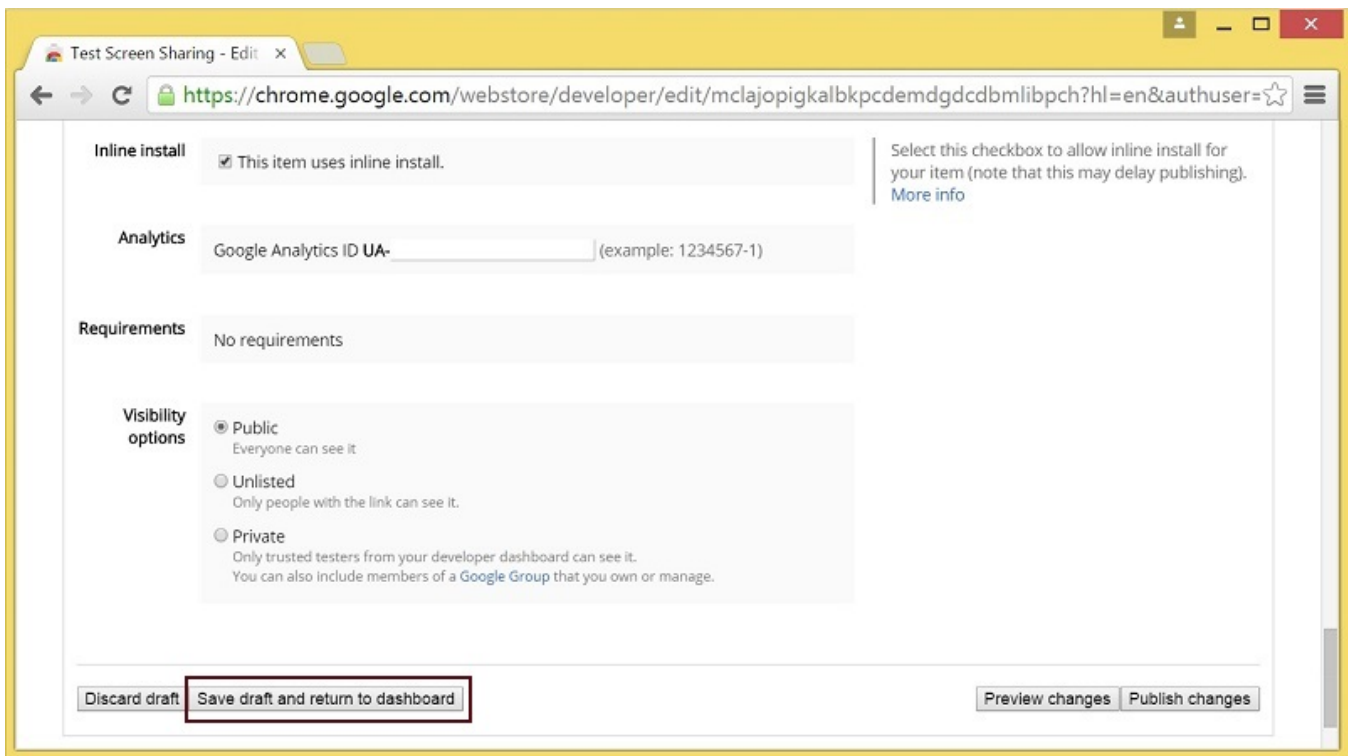


4. On the 'Upload' page, choose the chrome-extension.zip file and click 'Upload' button

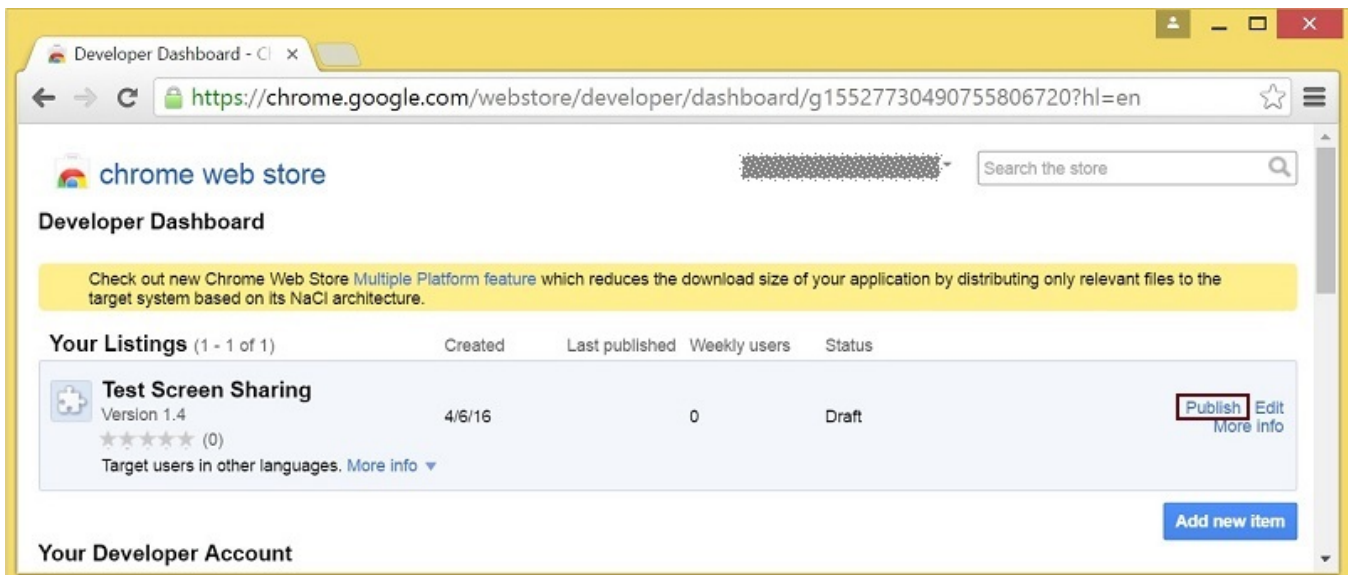


5. When the extension is uploaded, the page for editing the extension draft will be opened
Edit the extension as required and click 'Save draft and return to dashboard' button at the bottom of the page.

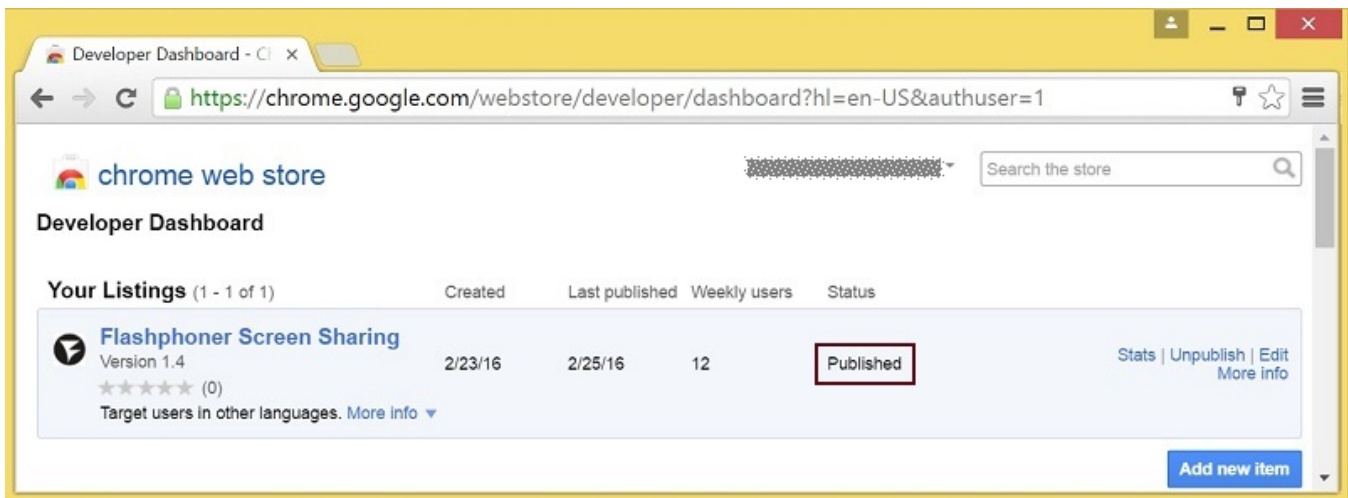




6. The extension will appear in the developer dashboard
Click 'Publish' link to publish the extension.



A published extension will have status 'Published' as on the image below.



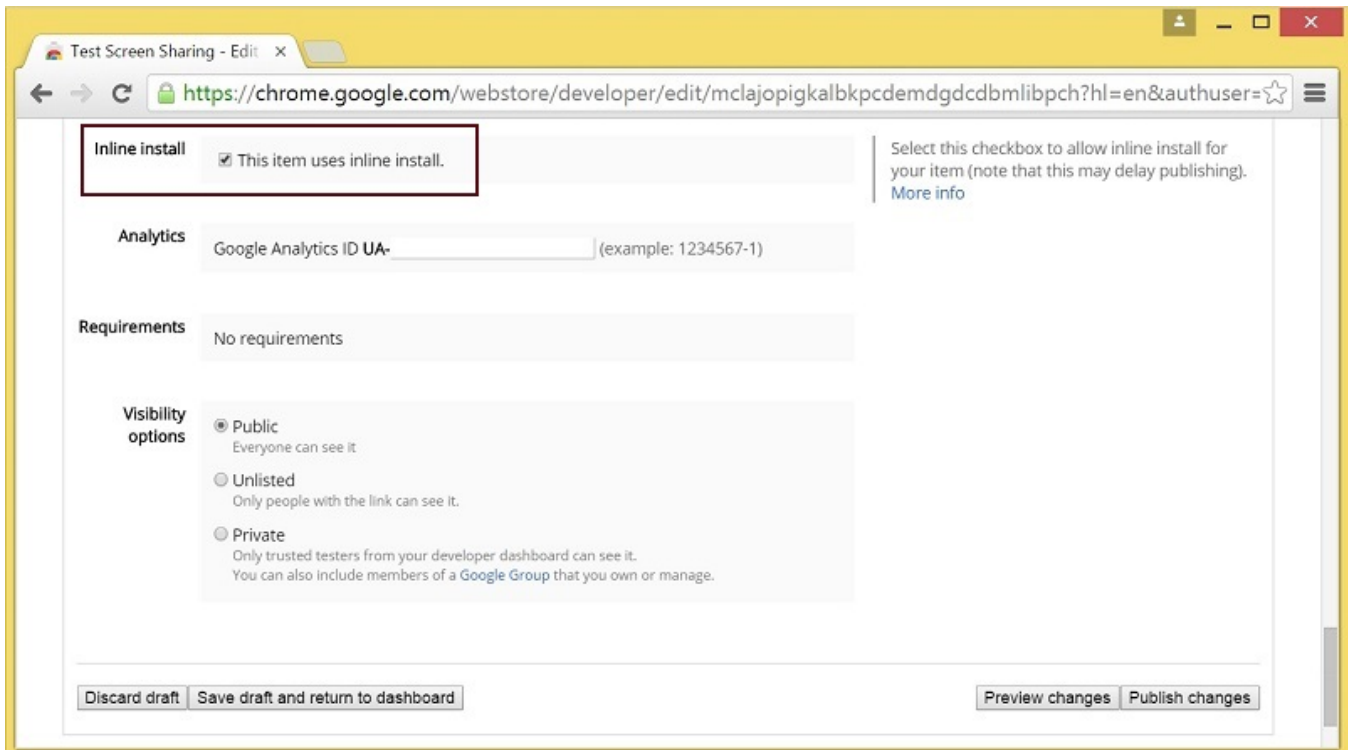
For more information, see [Chrome Web Store Publishing Tutorial](#).

Extension Inline Installation

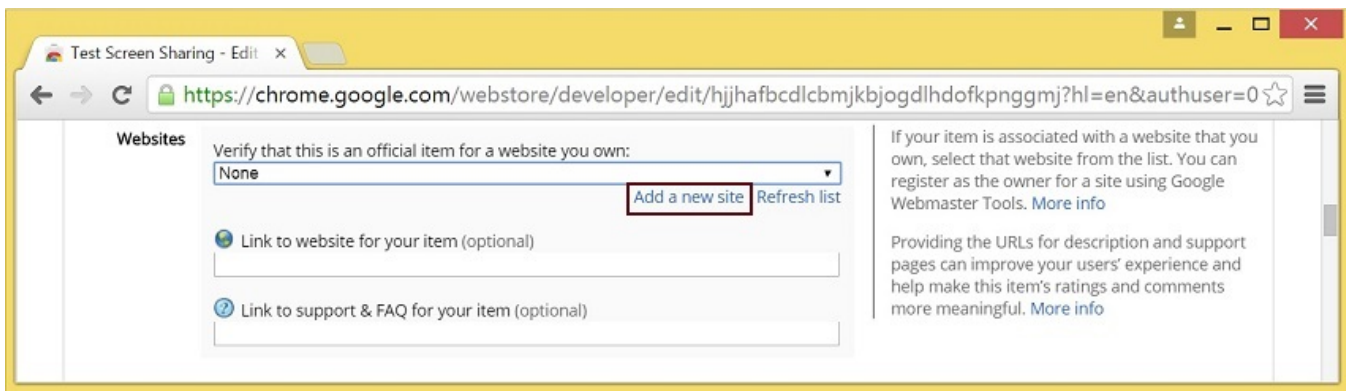
Extension installation can be initiated by clicking a link on the Screen Sharing client page. Note that the extension to be installed should be customized, published and approved before inline installation can be used.

Follow the procedures described below to use the client with your extensions.

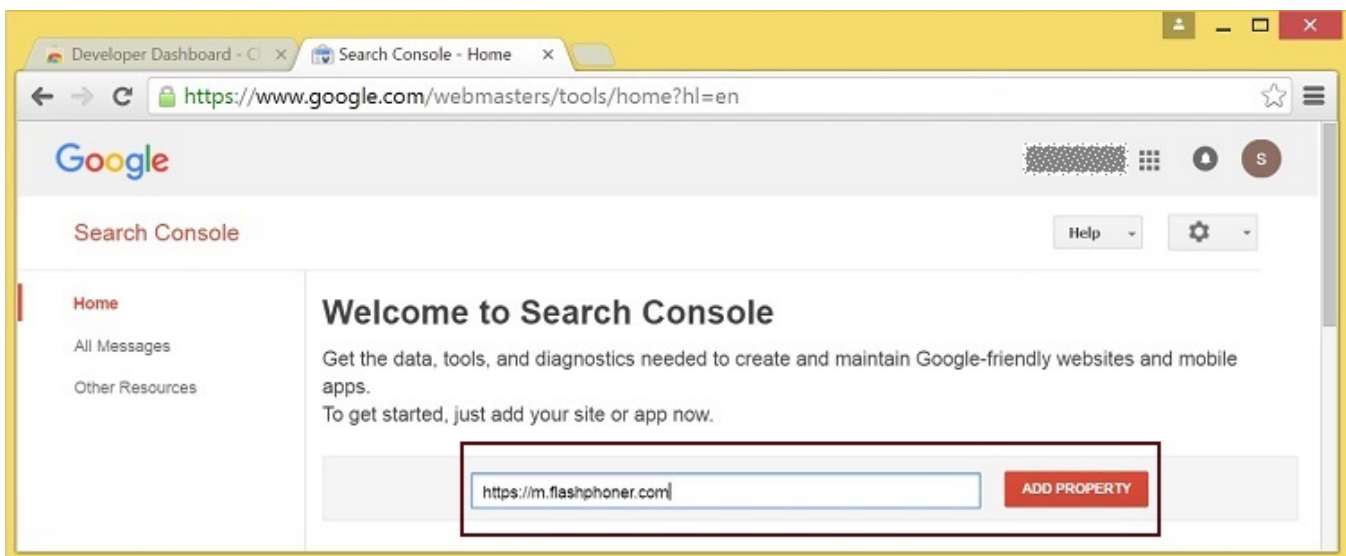
1. When publishing, select 'Inline Install' option



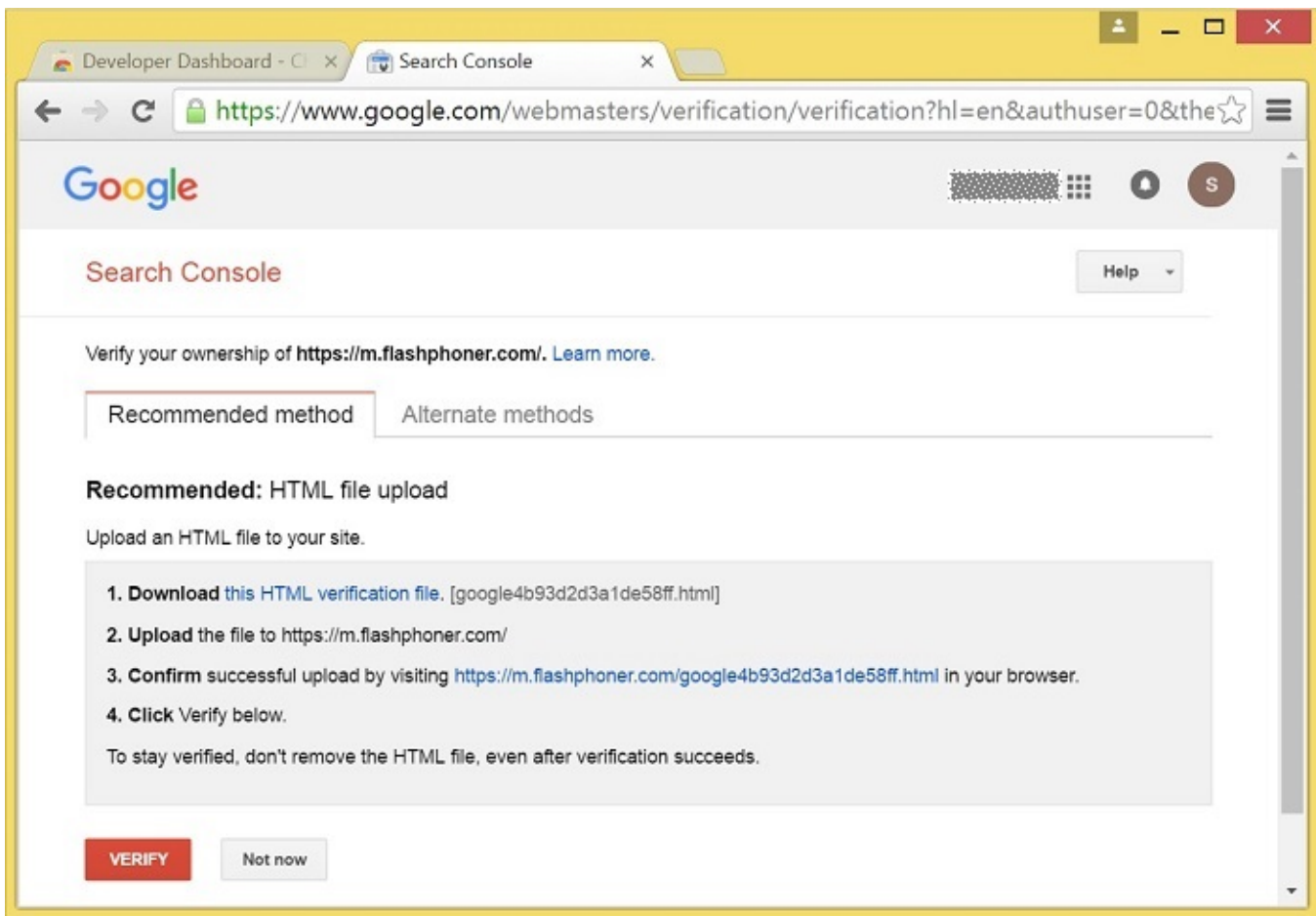
2. Verify and add website with your domain to the extension
- Click 'Add a New Site'



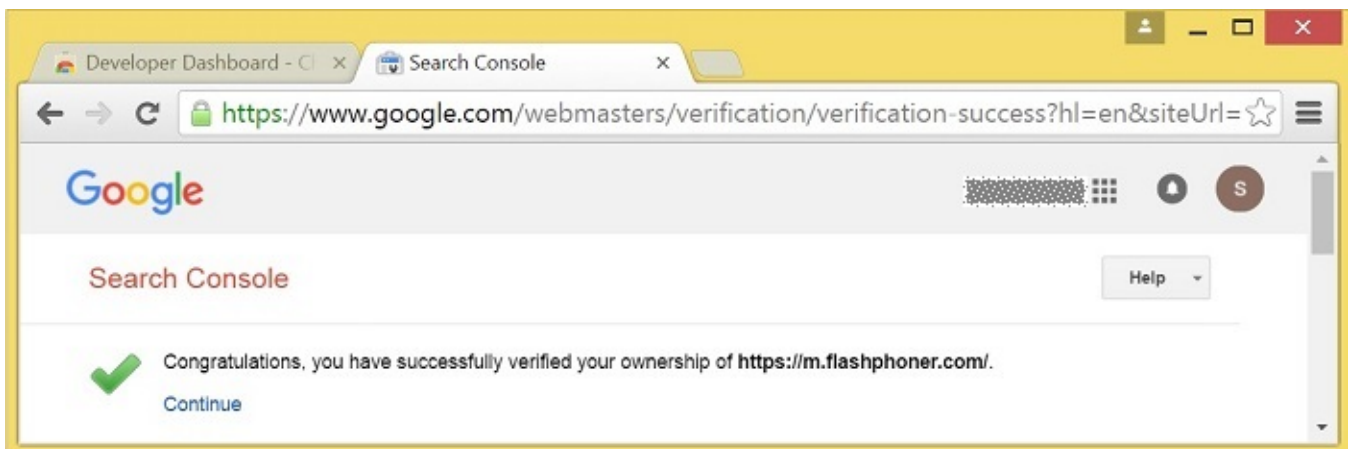
- Google Search Console page will open in a new tab
Enter URL with your domain.



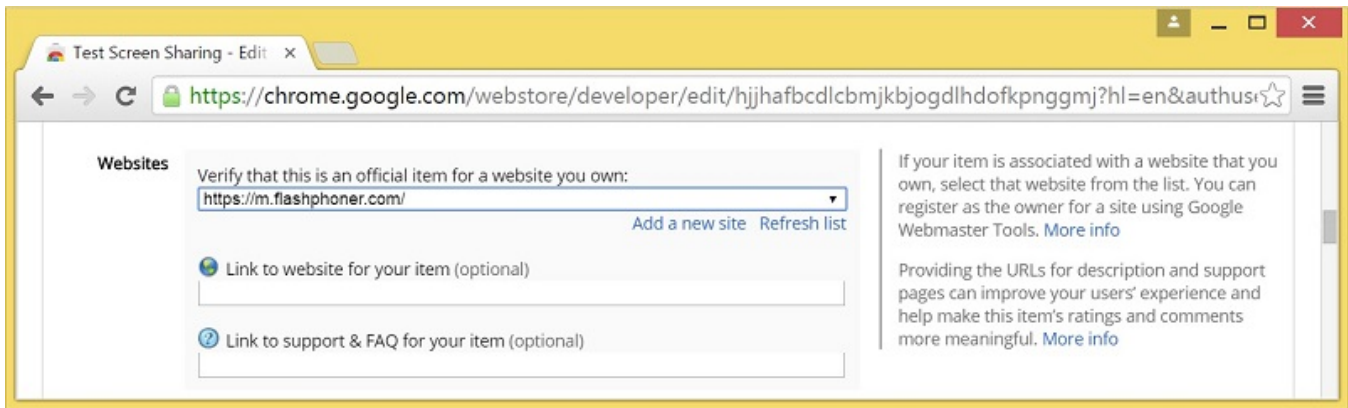
- A page with the instruction for the site verification will open
Follow the procedure steps and click 'Verify' button.



- If the verification is passed, a page confirming successful verification will be opened



- The website will appear in the list in the extension options and the extension can be associated with the site

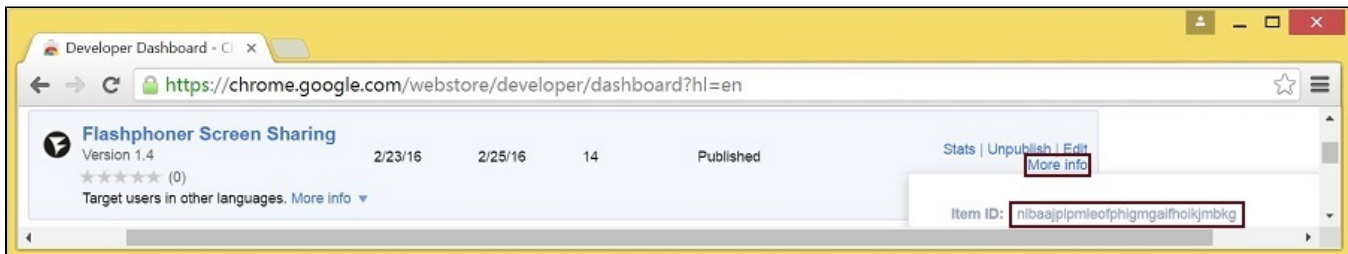


Configuring the client

Edit Screen-sharing.html and Screen-sharing.js

- In Screen-sharing.html the chrome-webstore-item parameter should point to your extension in the Chrome Store
- In Screen-sharing.js replace the value of the chromeScreenSharingExtensionId parameter to the ID of your extension

To get the ID of the extension, click 'More info' of this extension in [Chrome Developer Dashboard](#).



Media source parameters

To configure screen media source parameters, use parameters of the Configuration object passed to the init() method upon initializing of the Flashphoner API instance.

```
var f = Flashphoner.getInstance();
var configuration = new Configuration();
....
configuration.screenSharingVideoWidth = 1920;
configuration.screenSharingVideoHeight = 1080;
configuration.screenSharingVideoFps = 10;
f.init(configuration);
```

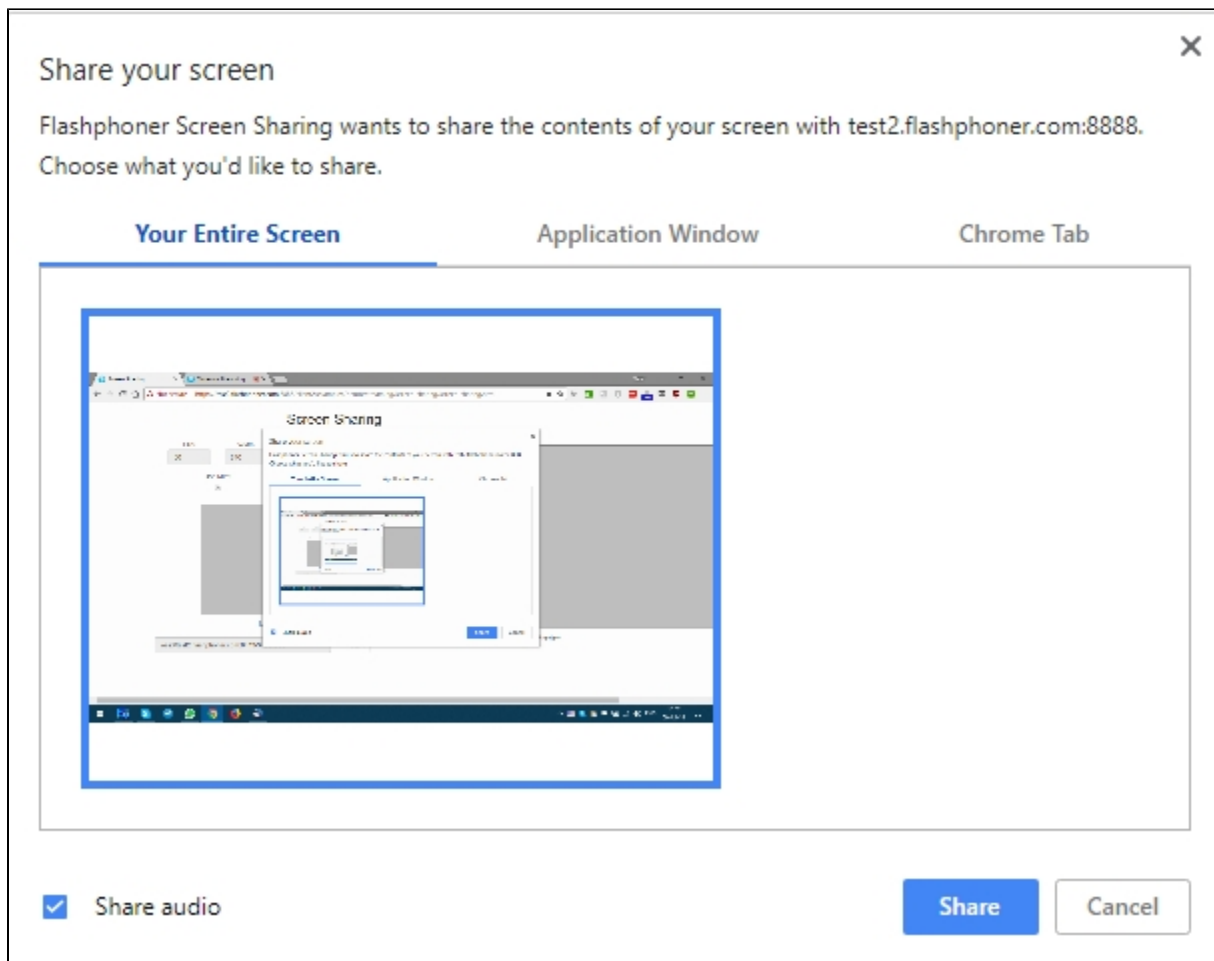
Parameter list

Parameter	Description
screenSharingVideoWidth	Screen media source width
screenSharingVideoHeight	Screen media source height
screenSharingVideoFps	Screen media source framerate

These parameters set marginal values of resolution and framerate (FPS). For instance, screenSharingVideoWidth = 1080 means the width of the source video cannot be more than 1080 pixels, but can be less. (i.e. when sending a stream of an app window that has the width of 720 pixels).

System sound capture in Chrome browser

In Chrome browser, there is ability to translate audio stream from system sound source while capturing a screen. The feature is useful in screencasting, for example. To capture system sound, set "Share audio" option in Chrome extension dialog window while choosing streaming source window or browser tab:



Chrome extension [code](#):

```
callback({sourceId: sourceId, systemSoundAccess: opts.canRequestAudioTrack});
```


Capture source (screen or window) management in Firefox browser

In Firefox browser, all the screen or some program window can be chosen as video stream source with `constraints.video.mediaSource` parameter

[code](#):

```
constraints.video.type = "screen";
if (Browser.isFirefox()){
    constraints.video.mediaSource = $('#mediaSource').val();
}
session.createStream({
    name: streamName,
    display: localVideo,
    constraints: constraints
})
```

Source selection interface example:

FPS	Width	Height	Source
30	640	480	Screen
Use Mic?		Mic	Screen
<input type="checkbox"/>		default: Microphone (Realtek)	Window
			
My Screen			
<input type="text" value="wss://test2.flashphoner.com:8443/f5b78c77"/>			Start

Program window capture

Screen Sharing

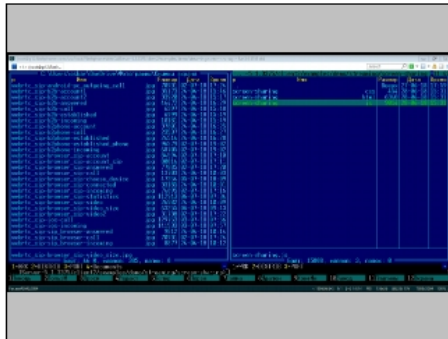
FPS Width Height Source
30 640 480 Window

Use Mic?

☐

Mic

default: Microphone (Realtek)

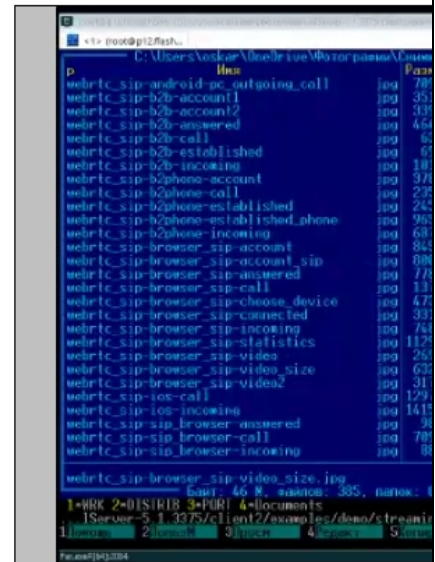


My Screen

wss://test2.flashphoner.com:8443/f5b78c77

Stop

PUBLISHING



Preview

Screen capture

Screen Sharing

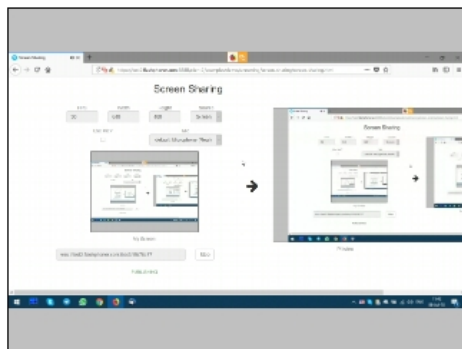
FPS Width Height Source
30 640 480 Screen

Use Mic?

☐

Mic

default: Microphone (Realtek)



My Screen

wss://test2.flashphoner.com:8443/f5b78c77

Stop

PUBLISHING



Preview

Screen sharing without extension

Firefox browser

Firefox browser does not use extension to share screen

Chromium based browsers

Since Chrome 73 and Flashphoner WebSDK [0.5.28.2753.86](#) screen sharing is possible without extension. To do this `constraints.video.withoutExtension` parameter should be passed when stream is created

[code](#)

```
if ($("#woChromeExtension").prop('checked')) {  
    constraints.video.withoutExtension = true;  
}
```

Safari browser in MacOS

Since Safari 13 and Flashphoner WebSDK [0.5.28.2753.152](#) screen sharing is possible without extension. To do this `constraints.video.withoutExtension` parameter should be passed when stream is created

[code](#)

```
if ($("#woChromeExtension").prop('checked') || Browser.isSafari()) {  
    constraints.video.withoutExtension = true;  
}
```

Known limits

1. In Chrome browser, picture resolution and FPS are set by source dimensions (screen, windows or browser tab) and by real picture updating speed, not by constraints/ This issue is fixed since Flashphoner WebSDK build [0.5.28.2753.152](#)
2. System sound capture only works starting from Chrome 74

Known issues

1. If the web application is inside an iframe element, publishing of the video stream may fail.

Symptoms: IceServer errors in the browser console.

Solution: put the app out of iframe to an individual page.

2. If publishing of the stream goes under Windows 10 or Windows 8 and hardware acceleration is enabled in the Google Chrome browser, bitrate problems are possible.

Symptoms: low quality of the video, muddy picture, bitrate shown in [chrome://webrtc-internals](#) is less than 100 kbps.

Solution: turn off hardware acceleration in the browser, switch the browser of the server to use the VP8 codec.

3. All streams captured from the screen stop if any one of them stops

Symptoms: while multiple streams are captured from the screen on one tab in the Chrome browser, if one stream is stopped, all streams stop.

Solution: cache tracks by the source of the video and stop them along with the last stream that uses that source, for example:

```

var handleUnpublished = function(stream) {
    console.log("Stream unpublished with status " + stream.status());
    //get track label
    var video = document.getElementById(stream.id() + LOCAL_CACHED_VIDEO);
    var track = video.srcObject.getVideoTracks()[0];
    var label = track.label;
    //see if someone using this source
    if (countDisplaysWithVideoLabel(label) > 1) {
        //remove srcObject but don't stop tracks
        pushTrack(track);
        video.srcObject = null;
    } else {
        var tracks = popTracks(track);
        for (var i = 0; i < tracks.length; i++) {
            tracks[i].stop();
        }
    }
    //release resources
    Flashphoner.releaseLocalMedia(streamVideoDisplay);
    //remove stream display
    display.removeChild(streamDisplay);
    session.disconnect();
};

```

4. Chrome browser stops sending video traffic when application window captured is minimized

Symptoms: if an application window captured is minimized to task bar, the stream freezes at subscribers side, and publishing may fail by RTP activity

Solution: in builds before [5.2.1784](#) disable RTP activity control for all the streams

```
rtp_activity_video=false
```

in builds since [5.2.1784](#) disable video RTP activity control by stream name template, for example

```
rtp_activity_video_exclude=.*-screen$
```