



Getting Started Guide for Cisco Jabber SDK for Web, Voice and Video Workload Version 4.0.0, System Release 11.0

First Published: December 19, 2013

Last Modified: July 08, 2015

Americas Headquarters

Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
<http://www.cisco.com>
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.

Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. and other countries. To view a list of Cisco trademarks, go to this URL: <http://www.cisco.com/go/trademarks>. Third-party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1110R)

© 2014 Cisco Systems, Inc. All rights reserved.



CONTENTS

CHAPTER 1	Preface 5 Cisco Jabber Voice and Video SDK 5
CHAPTER 2	Cisco Jabber Voice and Video SDK Overview 6 Architecture Overview 6
CHAPTER 3	Environmental Requirements 7 Supported Operating Systems and Browsers 7 Supported Processors 9 Supported Codec Versions 9 Supported CUCM and Expressway Versions 9 Cross-Origin Resource Sharing Support 9 Single Sign-On Requirements 11 Supported Identity Providers 11 SSO and Remote Access 11
CHAPTER 4	Getting Started 12 Naming Conventions 12 Add Browser-Based Softphone 13 Download the Cisco Web Communicator Add-on 13 Best Practices for Creating Applications for Chrome 13 Download SDK 14 Set Up Device Type in CUCM 14 Configure Device 16 Add Video Call Capabilities In-Browser 17 Introducing an in-browser-like experience for video conversations in Chrome browsers – Video Overlay feature 18 Add Video Call Capabilities in an External Video Window 19
CHAPTER 5	Feature Notes 21 Access Control Dialog Feature Overview 21 Access Control Dialog Box 22 Delayed User Authorization 22

Domain Whitelist Support	24
Language Support	25
Device Selection Feature Overview	26
Retain position of external video window	26
Certificate Validation Feature Overview	26
Service Discovery and Sign-in Types	27
Jabber SDK Discovery Based Sign-In Lifecycle	28
Single Sign On	30
Understanding Redirects and Same-Origin Issues	30
Setup Jabber SDK to use SSO feature	30
Device Selection	32
Volume control on Windows platforms	32
Custom ringtones on Windows platforms	33
Node Component	33
Install the Node Component	34
Node Component Configuration	35
Run the Node Component	36
Node Component Modules	37
Directory Navigator Sample Application	41
Set Up Directory Navigator	43

CHAPTER 6

Troubleshooting 44

Debugging JavaScript	44
Chrome	44
Firefox	44
Internet Explorer	44
Safari	44
JavaScript Events for Microsoft Windows Video Window	45
Disable App Nap	45
Common Errors	45
Uninstall Cisco Web Communicator Add-on	46
Log Files	46



CHAPTER 1

Preface

The Cisco Jabber Voice and Video SDK (part of the Cisco Jabber SDK for Web) allows you to embed voice and video telephony into your web pages. Although the SDK is designed to hide as much of the underlying complexity as possible, this guide is designed to give you important background information that will help you to get started as well as to understand some of the more advanced features of the SDK.

- [Cisco Jabber Voice and Video SDK, page 5](#)

Cisco Jabber Voice and Video SDK

With the Cisco Jabber Voice and Video SDK, you can enable the following telephony features directly in the browser:

- Make and receive calls
- Transmit and receive HD video
- Transfer calls
- Create and manage conferences
- Hold and resume calls
- Mute/unmute calls
- Use DTMF
- Control your desk phone

The Voice and Video SDK provides two phone operating modes:

- Desk phone mode: Control of a physical desk phone, including the ability to make calls, end the call, and use mid-call features
- Softphone mode: Softphone features, including desktop media termination and calling features.



CHAPTER 2

Cisco Jabber Voice and Video SDK Overview

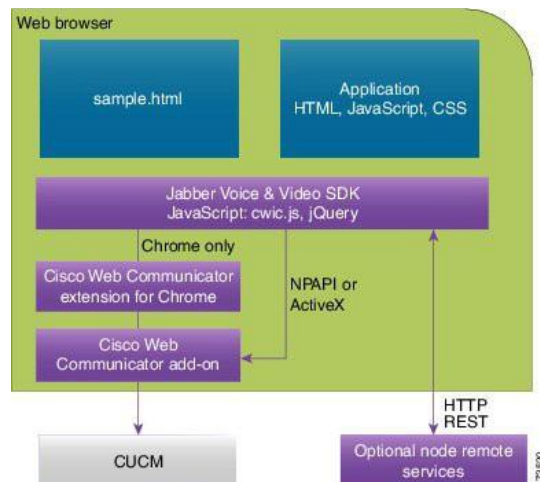
- [Architecture Overview, page 2](#)

Architecture Overview

The SDK itself contains the following components:

- Cisco Web Communicator add-on that provides call control and media termination within the browser.
 - Available on Windows and Mac operating systems.
 - Compatible with Internet Explorer, Firefox, Chrome and Safari.
 - Support for *soft phone mode* (connects SIP with Unified Communications Manager, and uses the computer for audio) and *desk phone mode* (connects CTI with Unified Communications Manager, and uses a physical phone for audio).
- JavaScript API enables Cisco voice and video telephony from a web page.
 - The cwic jQuery plug-in accesses telephony features.
- Optional server-side node that provides advanced features such as directory integration.

The following diagram shows the high-level architecture of the SDK.





CHAPTER 3

Environmental Requirements

The following sections describe the environmental requirements for Cisco Jabber SDK.

- [Supported Operating Systems and Browsers, page 7](#)
- [Supported Processors, page 9](#)
- [Supported Codec Versions, page 9](#)
- [Supported CUCM and Expressway Versions, page 9](#)
- [Single Sign-On Requirements, page 11](#)

Supported Operating Systems and Browsers

Windows

Browser		Windows 7	Windows 8	Windows 8.1
		32-bit and 64-bit		
Chrome: Stable channel 43	32-bit	X	X	X
	64-bit	X	X	X
Firefox: Final release channel 39	32-bit	X	X	X
	64-bit			
Internet Explorer 11	32-bit	X		X
	64-bit	X		X
Internet Explorer 10	32-bit	X	X	
	64-bit		X	

Browser		Windows 7	Windows 8	Windows 8.1
		32-bit and 64-bit		
Internet Explorer 9	32-bit	x		
	64-bit			
Internet Explorer 8	32-bit	x		
	64-bit			



Important

- Web browsers using Jabber SDK on Windows 8 and later must run in Windows desktop mode. For IE, adding a *requiresActiveX* "META" tag to your page will prompt users to reload IE in the traditional desktop mode if they are using IE in the new Windows UI.
- Using the SDK and Protected Mode in IE is not supported.
- Windows RT 8 and Windows RT 8.1 are not supported.

Mac OS X

Browser	Mac OS X 10.8 Mountain Lion	Mac OS X 10.9 Mavericks	Mac OS X 10.10 Yosemite
Chrome: Stable channel 43	x	x	x
Firefox: Final release channel 39	x	x	x
Safari 8			x
Safari 7		x	
Safari 6	x		

We recommend that App Nap, a feature in Mac OS X, be disabled for all browsers supported by the Jabber SDK.

Additional Notes

The Cisco Jabber SDK does not support virtual desktop environments such as Citrix XenApp, Remote Desktop Protocol, and so on.

Ensure that your browser is not configured to prevent browser plug-ins from running. In Internet Explorer browser plug-ins are referred to as *ActiveX Controls*. Chrome and Firefox use the term *plug-in*.

Supported Processors

Minimum CPU instruction set is SSE3.

Minimum CPU speed and type:

- Mobile AMD Sempron Processor 3600+ at 2 GHz
- Intel Core 2 CPU T7400 at 2.16 GHz

Supported Codec Versions

- G.711 a-law
- G.711 u-law
- G.722.1 24k
- G.722.1 32k
- G.729A

Supported CUCM and Expressway Versions

Cisco Unified Communications Manager 8.6(2) and later

Cisco Expressway Series for Cisco Unified Communications Manager

- Cisco Expressway-E, version 8.1.1 or later
- Cisco Expressway-C, version 8.1.1 or later



Note

The Cisco Jabber Voice and Video SDK does not support Cisco Unified Survivable Remote Site Telephony (SRST).

Cross-Origin Resource Sharing Support

In Cisco Unified Communications Manager 10.5, the Cross-Origin Resource Sharing (CORS) feature allows a web application to make an **XMLHttpRequest** to a domain other than the domain that the JavaScript originated from. Without CORS, such cross-domain requests would be forbidden by web browsers.

Using Cisco Unified Communications Manager User Data Service, an application now has the ability to get and change its own user data, such as getting a phone book or directory number (DN), or it can add a user to its phone book.

Cisco Unified Communications Manager Administration gives administrators the ability to add or remove domains to the access list. A domain can have Read-Only or Full-Access privilege:

- A Read-Only domain can issue an **HTTP GET** or **HEAD** command to the Cisco Unified Communications Manager data.
- A Full-Access domain can issue an **HTTP GET**, **HEAD**, **PUT**, **POST**, or **DELETE** command to the Cisco Unified Communications Manager data.

The following procedure describes how to add a domain for CORS access:

Procedure

- Step 1** Sign in to Cisco Unified Communications Manager Administration.
 - Step 2** Choose System > Cross-Origin Resource Sharing (CORS).
 - Step 3** Click Add New.
-

Sample code for a GET request

```
$.ajax({  
    url: encodeURI(url.value),  
    type: 'GET',  
    contentType: 'application/json',  
    dataType: 'json',  
    xhrFields: {  
        withCredentials: true  
    },  
    crossDomain: true,  
    success: function (data) {  
        status.value = JSON.stringify(data);  
    },  
    error: function () {  
        status.value = 'request failed';  
    }  
});
```

Sample code for a POST request

```
$.ajax({  
    url: encodeURI(url.value),  
    type: POST,  
    data: body.value  
    contentType: 'application/json',  
    dataType: 'json',  
    xhrFields: {  
        withCredentials: true  
    },  
    crossDomain: true,  
    success: function (data) {  
        status.value = JSON.stringify(data);  
    },  
    error: function () {  
        status.value = 'request failed';  
    }  
});
```

Single Sign-On Requirements

- Cisco Unified Communications Manager release 10.5.0 or later
- Configured redirect URI on CUCM
- Provide page at "redirect URI" location to get a token.

Supported Identity Providers

The IdP must be Security Assertion Markup Language (SAML) compliant.

- Ping Federate 6.10.0.4
- Microsoft Active Directory Federation Services (ADFS) 2.0
- Open Access Manager (OpenAM) 10.1

When you configure the IdP, the configured settings impact how you sign into the client. Some parameters, such as the type of cookie (persistent or session), or the authentication mechanism (Kerberos or Web form), determine how often you have to be authenticated.

SSO and Remote Access

- Cisco Unified Communications Manager release 10.5.2 or later.
- Cisco Expressway 8.5
- The Identity Provider used must have the same internal and external URL. If the URL is different, the user may be prompted to sign in again when changing from inside to outside the corporate firewall and vice versa.



CHAPTER 4

Getting Started

- [Naming Conventions, page 12](#)
- [Add Browser-Based Softphone, page 13](#)
- [Add Video Call Capabilities In-Browser, page 17](#)
- [Introducing an in-browser-like experience for video conversations in Chrome browsers – Video Overlay feature, page 18](#)
- [Add Video Call Capabilities in an External Video Window, page 19](#)

Naming Conventions

With Google's deprecation of the Netscape Plug-in Application Programming Interface (NPAPI) in Chrome, the architecture for the Jabber SDK to work with Chrome required a change from an NPAPI plug-in for Chrome to both a Chrome extension and a native-OS add-on to Chrome. These new components are in addition to the existing plug-ins for other supported browsers. Terminology changes were needed as follows:

- The term *add-on* is used as both:
 - The general name for installed software used in addition to a browser
 - The specific software used in addition to Chrome
- The term *plug-in* is now only used for the installed software used in addition to Internet Explorer, Firefox, and Safari.
- The installers for Windows and Mac were renamed to reflect installing both of the following:
 - The NPAPI and ActiveX plug-ins for non-Chrome browsers
 - The add-on for Chrome

The new installers are named *CiscoWebCommunicatorAddOnInstaller.exe* and *CiscoWebCommunicatorAddOnInstaller.dmg*.

- The existing plug-in for Internet Explorer, Firefox, and Safari is named the *Cisco Web Communicator plug-in*.
- The new components for Chrome support are the Cisco Web Communicator extension for Chrome and the Cisco Web Communicator add-on for Chrome. The Cisco Web Communicator add-on for Chrome is what Chrome calls a native messaging host.

Add Browser-Based Softphone

The Cisco Jabber SDK works with Cisco Unified Communications Manager software to provide voice and video functionality. You will either need access to your own Cisco Unified Communications Manager infrastructure, or you can check our testing overview for other options.

See the steps in the following sections to configure a browser-based softphone.

Download the Cisco Web Communicator Add-on

The Cisco Web Communicator add-on is required for signaling and media termination in the browser through `cwic.js`. It contains the:

- Cisco Web Communicator plug-in (Netscape Plug-in Application Programming Interface [NPAPI] for Firefox and Safari; ActiveX for Internet Explorer).
- Cisco Web Communicator add-on for Chrome.

To use the SDK in Chrome, the Cisco Web Communicator extension for Chrome is also required. The extension is a bridge between `cwic.js` and the Cisco Web Communicator add-on for Chrome.

The Cisco Web Communicator extension for Chrome and the Cisco Web Communicator add-on for Chrome are required because Google will no longer support the NPAPI, which is what permitted direct communication between the web page and the Cisco Web Communicator plug-in (NPAPI) for Chrome used in earlier versions of the Jabber SDK.

Procedure

-
- Step 1** Download and install the Cisco Web Communicator add-on.
If you have already purchased a user license for the Jabber SDK (meaning that you have purchased UCL, CUWL or other relevant licensing for the target users, along with the appropriate support and upgrade contract, for example, ESW and UCSS), download the add-on from the following location: <http://software.cisco.com/download/type.html?mdfid=283882159&catid=null>.
- Step 2** If you want to use the SDK in Chrome, download and install the Cisco Web Communicator extension for Chrome from the Chrome Web Store: <https://chrome.google.com/webstore/detail/cisco-web-communicator/pbblmlcmhfnfflbkbinnhacecaankdh>
-

Best Practices for Creating Applications for Chrome

Obtaining the Add-on and Extension

The Chrome Jabber SDK implementation requires users to install both the Cisco Web Communicator add-on and the Cisco Web Communicator extension for Chrome.

To make it as easy as possible for Chrome users to obtain the extension and the add-on, provide the following in your application:

- A link to the extension in the Chrome Web Store.
- A link to the location where you distribute the add-on.

We suggest that you set the links to open in a new window or tab so that users do not navigate away from your application. Here is some sample code for handling the missing extension. The `target="_blank"` attribute causes the link to open in a new window or tab:

```
function phoneErrorCallback(error, exception) {
    if (error) {
        settings.log('phone error: ', error);

        var msg = 'ERROR: cannot initialize phone: ' + error.message + ' (code ' +
            error.code + ')<br>';

        if(error.code == 'ExtensionNotAvailable'){
            msg += '\nChrome extension available at <a href="https://chrome.google.com/
                webstore/detail/cisco-web-communicator/ppbllm1cmhfnfflbkbinhacecaankdh/"
                target="_blank" style="color:blue">chrome web store</a>.';
        }

        $('#msgcontainer').empty().html('<b style="color:red">' + msg + '</b>');

        var about = $().cwic('about');
        showVersions(about);
    }

    if (exception) {
        settings.log('exception: ',exception);
    }
};
```

Permissions Requested

The following tables describes the permissions requested by the Cisco Web Communicator extension for Chrome:

Permission Requested	Reason
Access to your data on all websites	Allows the extension to inject, into every page, a content script that allows cwic.js to communicate with the extension.
Manage your apps, extensions, and themes	Allows the extension to determine the host name of a web page attempting to use cwic.js. This is the name you authorize when the add-on displays an Access Control Dialog (ACD).
Communicate with cooperating native applications	Necessary for the extension to communicate with the Cisco Web Communicator add-on.

Download SDK

Download the Cisco Jabber Voice and Video SDK. You must run the SDK and Sample App contained therein on a web server while you develop and test your code. Running from the local file system is an unsupported configuration, and not all functionality may work.

Set Up Device Type in CUCM

This section explains how to set up the correct device type in Cisco Unified Communications Manager to enable you to use the Cisco Jabber SDK. In brief, it involves creating a CSF device with the name ECP<yourusername>, but follow these detailed steps to ensure proper functionality.

Procedure

- Step 1** Sign in to Cisco Unified Communications Manager Administration.
- Step 2** Choose **Device > Phone**.
The Find and List Phones window opens.
- Step 3** Click **Add New**.
The Add a New Phone window opens.

- Step 4** From the **Phone Type** drop-down list, select **Cisco Unified Client Services Framework**.
- Step 5** Click **Next**.
The Phone Configuration window opens.
- Step 6** In the **Device Information** section of the Phone Configuration window, set the following:
- **Device Name**—By default, the name should be of the form: ECP<username>. Example: ECPjohndoe. However, the SDK allows you to override the prefix to one of your own by passing in the devicePrefix field in the settings object. You can also use your own custom scheme for associating CUCM device names with usernames by providing your own settings.predictDevice function.
 - The device name is not case sensitive.
 - The device name is created by placing the prefix in front of the username and then removing any characters which are not permitted. Symbols such as dots, hyphens, underscores must be stripped, as well as any accented characters or characters not in the Latin (English) alphabet.
 - Cisco Unified Communications Manager accepts a maximum length of 15 characters, so the generated name must be truncated to this length.
 - This may lead to some name clashes because names that are only unique in the thirteenth character and beyond become the same name when ECP is prepended and the total length is truncated to 15. Also, the names Joe.Bloggs and JoeBloggs both map to the same device name: ECPJoeBloggs. These ambiguities must be handled on a case-by-case basis, and may require that the user names be changed to make them unique.
 - Similarly, the user Frédéric will have a device name of ECPFrdric. Dropping the non-Latin characters can lead to further name clashes.
 - **Description**—Enter a descriptive name, such as John Doe's Web Communicator.
 - **Device Pool**—Set to the desired device pool.
 - **Phone Button Template**—Set to **Standard Client Services Framework**.
- Step 7** In the Protocol Specific Information section of the Phone Configuration window, set the following:
- **Device Security Profile**—Set to **Cisco Unified Client Services Framework - Standard SIP Non-Secure**.
 - **SIP Profile**—Set to **Standard SIP Profile**.
- Step 8** Click **Save**.
- Step 9** Click **Apply Config** if this button is available (and confirm when prompted).
- Step 10** To add a line for the Cisco Web Communicator device, click **Line [1] - Add a New DN** on the upper-left portion of the Phone Configuration window.
The Directory Number Configuration window opens.
- Step 11** Type in a Directory Number.
- Step 12** Click **Save**, and then click **Apply Config** if this button is available (and confirm when prompted).
You should see the name of the softphone device in the Associated Devices box on the Directory Number Configuration page.
- Step 13** Scroll down to the bottom of the Directory Number Configuration page. Click **Associate End Users**.
The Find and List Users window opens.
- Step 14** Use the search criteria to find the user you want to associate with the directory number, then check the box next to that user name and click **Add Selected**.
The Directory Number Configuration window should now show that the user is associated with the line. This information appears near the bottom of the window in the section called **User Associated With Line**.

- Step 15** Click on the user name in the **User Associated with Line** section of the window. The End User Configuration window opens.
- Step 16** Scroll down to the Direct Number Associations section of the window and select the primary extension from the Primary Extension drop-down list. This is the directory number you configured in step 11.
- Step 17** In the Permissions Information section at the bottom of the End User Configuration window, click Add to **User Group**. The Find and List User Groups window opens.
- Step 18** Use the search criteria to find Standard CCM End Users.
- Step 19** Check the box next to **Standard CCM End Users**, then click **Add Selected**. The Standard CCM End Users group should now appear in the Permissions Information section at the bottom of the End User Configuration window.
- Step 20** Click **Save**. The Cisco Web Communicator device is now configured in Cisco Unified Communications Manager.
-

Configure Device

Perform the following steps to configure your device using the SDK.

Procedure

- Step 1** Open sample.html that comes with the SDK download.
- Step 2** Enter your Cisco Unified Communications Manager username, password, and IP address in the fields provided.
- Step 3** Select **SoftPhone** as the mode.
- Step 4** Click **Login**.
- Step 5** Select the device and then click **Connect**. The **Make Call** button will be enabled once your soft phone registers with Cisco Unified Communications Manager.
- Step 6** Enter a number into the **Number** field and click **Make Call**.
- Step 7** Finally, locate a colleague with whom you can make test calls and initiate some browser-to-browser video calls. Alternatively, you can make calls to existing Cisco hardware and software voice and video products.
-

What to Do Next

Deploy the cwic.js and ciscobase.js files to your web server and add a softphone to your existing web application.

Add Video Call Capabilities In-Browser

This section demonstrates how to add video call capabilities to your web page.

You can display video in-browser in Mozilla Firefox, Microsoft Internet Explorer, and Apple Safari browsers.

Procedure

- Step 1** The first thing to do is complete the steps in *Add Browser-Based Softphone*. The initial setup steps for voice and video are identical.
- Step 2** After you have successfully made a voice call with the sample app (**sample.html**) that is included with the SDK, check the **'with Video'** checkbox and make a call to another video phone (you can do this between two computers using the SDK). The sample page shows you a window that contains the other party's video, as well as controls to display your own video feed.
- Step 3** The next step is to add this capability to your own page. As a start, you can use the code that was written for the *Embed a softphone in under 5 min* tutorial. The SDK requires a video object embedded in the page as a place to render the video stream. You can do this with the **createVideoWindow** function. This function can be called at any time after the page has loaded. In this example, it is called inside the **ready** callback.

```
$( '#container'
  ).cwic('
    init', {
  ready:
    function(
    ) {
      $('#remotevideocontainer').cwic('createVideoWindow',{ id:
        'videoobject',
        success: function(pluginid) {
          //Video object has correctly loaded
        }
      });
      $(this).cwic('registerPhone', {
        user: 'dave',
        cucm: '11.22.33.44',
        password: '',
        success: function() {
          //phone registered
        }
      });
    }
  });
```

This code initializes a video object in the page that can be used to display the video stream from the remote user once you are on a call.

- Step 4** The next step is to enable this video code once you make a call. The SDK allows you to do this once a conversation has started by using the **updateConversation** method, but this example shows how to add video from the very start of the call. You can do this easily by passing in two additional parameters to the **startConversation** method. This code is the click handler for the **Make Call** button. The **videoDirection** parameter is used to tell the softphone whether you want to send video, receive it, or both. In this case, both send and receive video are chosen for the best experience. The other parameter, **remoteVideoWindow**, is the ID on the page of the video window created earlier. This enables the SDK to render the video from the remote party on the screen.

```
$('#callbtn').click(function() {
  var num = $('#numtodial').val();
  $('#container').cwic('startConversation', {
```

```

        participant: {
        recipient: num },
        videoDirection:
        'SendRecv',
        remoteVideoWindow:
        'videoobject'
    });
});

```

The dimensions and appearance of the video window can be controlled using CSS.

Introducing an in-browser-like experience for video conversations in Chrome browsers – Video Overlay feature

Deprecation of NPAPI disabled the in-browser video for Chrome. In order to mitigate this issue, docking capabilities were added to the external video window, thus emulating in-browser video. In docked state, the external video window is overlaid on top of the Chrome browser, without displaying the window title. The developer should provide a target container (HTML element) above which the external video window will be overlaid. Size and position required for displaying the external video window over the targeted container are being calculated by the add-on and cwic. If the targeted container is moved or resized, that gets detected by cwic and the add-on, and the video overlay position is updated accordingly.

Docking capability usage examples:

- The following code snippet docks the external video window, created with *showPreviewInExternalWindow* or *showCallInExternalWindow* API functions, to the specified rectangular HTML element, most commonly a 'div' element. This is a simple case where the target video container is on the same HTML page as cwic.js file.

```
$('#videoContainerId').cwic('dock');
```

- The following code snippet docks the external video window to the specified rectangular HTML element, which resides on the specified window object (could be a window object of iframe or popup).

```

$().cwic('dock', {
    window: windowElementOfSomeHtmlDocument,
    element: targetElementForVideoOverlay
});

```

- The following code snippet undocks the external video window.

```

$('#videoContainerId').cwic('undock');
// or
$(document).cwic('undock');
// or
$().cwic('undock');

```

Limitations of the video overlay feature:

- Page title of the HTML page where the video window is meant to be displayed using docking feature should be unique among all other windows displayed on the desktop. Otherwise, video window could be displayed in an unexpected manner.
- When the video window container is moved, either by moving the browser application or dragging & dropping the video container, the video window is slightly lagging while movement is in progress.
- Implementation for OS X requires a unique target to be present on the screen which is used for detecting the correct position of the video window. Currently, this target is a small magenta (#FF00FF) colored square followed by black color. Considering this, the video window could be docked with an unexpected outcome

if any other element with a color similar to magenta is displayed on the page.

Related Topics

[Add Browser-Based Softphone, on page 13](#)

Add Video Call Capabilities in an External Video Window

You can use an external video window for video calls and to see a self-view. The video window has embedded **Mute** and **End** call buttons, similar to Cisco Jabber for Windows.

Supported Browsers

The external video window is available for all supported browsers:

- For Google Chrome browser, the external video window is the only way to provide video calling, with addition of video overlay feature which offers in-browser-like experience for video conversations. You must install the Cisco Web Communicator extension for Chrome, as well as the Cisco Web Communicator add-on. For more information, see *Download the Cisco Web Communicator Add-on*.
- For Mozilla Firefox, Microsoft Internet Explorer, and Apple Safari browsers, you can display video in an external video window or in-browser.



Tip

If your web app will be available to users who use Chrome, we recommend that you implement the external video window so that all users will have a consistent experience.

Features

The following UI controls overlap with functionality provided by the cwic API:

- Mute/Unmute microphone
- Start/Stop video
- A new keypad control that allows the user to send in call key presses
- End call

The following UI controls offer functionality not provided by the cwic API:

- Maximize/Minimize the window
- Resize the window
- Ability to make the window full screen

You can close the external video window directly from the window by clicking the **Close** button (the X) in the upper right of the window. Clicking the **Close** button will also end the call, if one is active for the external video window.

API

The following cwic methods provide external video window support:

Method	Description
<code>showCallInExternalWindow(callId)</code>	Shows the call in the external video window
<code>showPreviewInExternalWindow()</code>	Shows preview (self-view) in an external video window
<code>hideExternalWindow()</code>	Hides an external video window created by <code>showPreviewInExternalWindow</code> or <code>showCallInExternalWindow</code> Note Using the <code>hideExternalWindow</code> cwic API method does not do the same thing as clicking the Close button on the external video window. If a call is in progress, closes the window, but does not end the call. Clicking the Close button will close the window and end the call.
<code>setExternalWindowAlwaysOnTop(isAlwaysOnTop)</code>	Controls whether external video windows created by <code>showPreviewInExternalWindow</code> or <code>showCallInExternalWindow</code> are shown always on top (default) or not
<code>setExternalWindowShowSelfViewPip(showPipSelfView)</code>	Controls whether a picture-in-picture preview (self-view) is shown when <code>showCallInExternalWindow</code> is used to put a call in external video window
<code>setExternalWindowTitle(title)</code>	Sets the window title used in external video windows created by <code>showPreviewInExternalWindow</code> or <code>showCallInExternalWindow</code>
<code>getExternalWindowState()</code>	Triggers an <code>externalWindowEvent</code> to be sent to the application with the current state of the external window
<code>setExternalWindowShowControls()</code>	Controls whether overlaid controls are shown in external video window

The following cwic event provides external video window support:

Event	Description
<code>externalWindowEvent</code>	A property has changed on the external video window created by <code>showPreviewInExternalWindow</code> or <code>showCallInExternalWindow</code>



Feature Notes

This chapter provides information on select features of the Voice and Video SDK.

- [Access Control Dialog Feature Overview, page 21](#)
- [Device Selection Feature Overview, page 26](#)
- [Retain Position of External Video Window, page 26](#)
- [Certificate Validation Feature Overview, page 26](#)
- [Service Discovery and Sign-in Types, page 27](#)
- [Single Sign On, page 30](#)
- [Device Selection, page 32](#)
- [Volume Control on Windows Platforms, page 32](#)
- [Custom Ringtones on Windows Platforms, page 33](#)
- [Node Component, page 33](#)
- [Directory Navigator Sample Application, page 41](#)

Access Control Dialog Feature Overview

This feature is available for the Voice and Video SDK version 3.0.0 and later.

The SDK provides a GUI control, Access Control Dialog, for user notification and response when a website that has not been previously visited, or that is not in a whitelisted domain, tries to use the add-on. The dialog box is displayed in front of the browser. On some OS/browser combinations, the user may not be able to interact with the browser or interact with the webpage while the dialog box is open.

This feature is invoked when the add-on is initialized. It specifies the domain that is requesting permission and provides the user with the option to Deny, Allow or Always Allow access to a desktop phone, camera or microphone.

Always Allow selections are shared across all browsers on both Macintosh and Windows systems.

Access Control Dialog Box

Name	Description
User notification	"Allow the page to control your desk phone or to access your microphone and camera? Clicking Deny may limit application behavior during this session."
Always Allow	Website permitted to use the add-on during the current visit and for all future visits by the same user from the same client machine.
Allow	Website permitted to use the add-on during the current visit. Future visits to the same website trigger the modal dialog box.
Deny	Website not permitted to use the add-on during the current visit. Future visits to the same website trigger the modal dialog box.

Delayed User Authorization

In addition to providing the Access Control Dialog, the API provides methods to control when the dialog is displayed.

Methods and fields	Description
<code>getUserAuthStatus()</code>	Returns a String value that indicates the authorization status for the current user.
<code>showUserAuthorization(args)</code>	Show user authorization dialog. The <code>args.denied</code> parameter specifies a function to call if the user presses the Deny button on the user authorization dialog box. If the user presses the Allow or Always Allow button, the settings.ready callback (which was passed into init) will be called.
<code>settings.delayedUserAuth</code>	A callback used to indicate that CWIC must show the user authorization dialog before the application can use the CWIC API.
<code>about.capabilities.delayedUserAuth</code>	about is the object returned by <code>cwic about</code> . about.capabilities can contain a boolean delayedUserAuth property.

To enable the Delayed User Authorization functionality, an application must specify a callback function as the **delayedUserAuth** property in the **settings** object that is passed to the **init** method.

If the **delayedUserAuth** setting is false or not present, or if the add-on has been whitelisted for the site, behavior in regards to the authorization dialog is the same as in release 3.0.0.

If the **delayedUserAuth** callback is present, it will be called if the add-on has not yet been authorized for the site.

If the site was not previously authorized by the user or administrative whitelist, and the application provides a **delayedUserAuth** callback, the application must use the **showUserAuthorization** API and successfully authorize the site to the user before using the CWIC API.

The **showUserAuthorization** API will result in the user authorization dialog being shown. If the user chooses **Allow** or **Always Allow**, the **settings.ready** callback will be called as normal. If the user chooses **Deny**, the denied callback provided to the **showUserAuthorization** API will be called, and the application should not use the CWIC API or add-on. Once the user has denied authorization, most CWIC API calls will result in the error **NotUserAuthorized**.

Example

To delay showing the access control dialog, use a function like the following:

```
delayedUserAuth parameter to cwic('init');
```

```
function delayedUserAuthCallback() {
    // Indicate that the add-on is not authorized.
    $('#auth-status').text('Add-on not authorized');
}
```

Then after the **delayedUserAuth** callback has been made, call **cwic('showUserAuthorization')** with the denied callback as one of its parameters. This could be done in response to a button click, a timer, or some other user interaction. For example:

```
$('#auth-button').click(function() {
    $('#container').cwic('showUserAuthorization', {
        denied: function() {
            alert('Add-on authorization denied by user.');
        }
    });
});
```

Once **cwic('init')** has been called, you may call **cwic('about')** at any point to determine version information of the add-on. No other CWIC API calls should be made until either the **'ready'** or **'delayedUserAuth'** callback has been called. The callback is the indication from the application that the asynchronous add-on load is complete.

Domain Whitelist Support

An administrator is able to add domains to the whitelist. The add-on install process does not modify existing whitelist entries.

Mac OS X.x

On OS X.x, an administrator adds the whitelisted domains to the add-on preference list:

`com.cisco.CiscoWebCommunicator.plist`

For example, to whitelist `jabberweb.cisco.com`, create the Key `AlwaysAllow_jabberweb.cisco.com` and set its Boolean value to `Yes`.

For user specific preferences:

- On Mac OS X 10.7, 10.8 and 10.9, add `com.cisco.CiscoWebCommunicator.plist` to the `~/Library/Preferences/folder`. Overwriting this file may remove existing entries.
- On Mac OS X 10.10, run the following command:
`defaults import com.cisco.CiscoWebCommunicator`
`/path_to_file/com.cisco.CiscoWebCommunicator.plist`

For system-wide preferences, add `com.cisco.CiscoWebCommunicator.plist` in the root `HD/Library/Preferences/folder`.

Windows

In order to whitelist a domain, create a DWORD registry value under the following keys with the specified parameter values.

User whitelist

Key: `HKEY_CURRENT_USER\SOFTWARE\Cisco Systems, Inc.\Web Communicator\AlwaysAllow`

Domain to whitelist: `jabberweb.cisco.com`

Boolean value: `1`

Administrator whitelist

Key: `HKEY_CURRENT_USER\SOFTWARE\Policies\Cisco Systems, Inc.\Web Communicator\AlwaysAllow`

Domain to whitelist: `jabberweb.cisco.com`

Boolean value: `1`



Note

The Windows add-on merges the administrator and user-allowed domains by reading these two registry keys

As of 3.0.1:

Under the admin whitelist, administrators can use either key to push whitelisted domains out to users.

Administrator whitelist:

`HKEY_CURRENT_USER\SOFTWARE\Policies\Cisco Systems, Inc.\Web Communicator\AlwaysAllow`
`HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Cisco Systems, Inc.\Web Communicator\AlwaysAllow`



Note

Whitelisting is intended for managed environments where registry changes can be pushed out through some standard process.

Language Support

The Access Control Dialog box supports localization of the user notification message.

Languages

- Arabic (Saudi Arabia)
- Bulgarian (Bulgaria)
- Catalan (Catalan/Spain)
- Chinese (China)
- Chinese (Taiwan)
- Croatian (Croatia)
- Czech (Czech Republic)
- Danish (Denmark)
- Dutch (Netherlands)
- English (United States)
- Finnish (Finland)
- French (France)
- German (Germany)
- Greek (Greece)
- Hebrew (Israel)
- Hungarian (Hungary)
- Italian (Italy)
- Japanese (Japan)
- Korean (Korea Republic)
- Norwegian Bokmal (Norway)
- Polish (Poland)
- Portuguese (Brazil)
- Portuguese (Portugal)
- Romanian (Romania)
- Russian (Russia)
- Serbian (Republic of Montenegro)
- Slovak (Slovakia)
- Slovenian (Slovenia)
- Spanish (Spain)
- Swedish (Sweden)
- Turkish (Turkey)
- Thai (Thailand)

Device Selection Feature Overview

The SDK provides means to choose from all available cameras, microphones, speakers, and other audio devices. This feature is available for the Voice and Video SDK version 3.0.0 and later. The device selection API consists of the following event and functions:

Name	Description
<code>mmDeviceChange.cwic</code>	This event will be sent by CWIC whenever a change to the list of devices is detected. The application should use <code>getMultimediaDevice()</code> to acquire the updated list of devices.
<code>getMultimediaDevices()</code>	A function that retrieves a list of objects describing the multimedia devices installed on a system.
<code>setRecordingDevice()</code>	A function that sets the audio recording device used by the add-on.
<code>setCaptureDevice()</code>	A function that sets the video capture device used by the add-on.
<code>setPlayoutDevice()</code>	A function that sets the play-out device used by the add-on.

Retain position of external video window

The position and size of the undocked external video window is retained between calls, browser restarts, different browsers or machine reboots. While the external video window is docked, the position and size of window are not saved. After undocking the window will retain its last position and size prior to docking.

The information about window position and size is kept in Registry on Windows OS and Preferences file on Mac OS.

Win: HKEY_CURRENT_USER\SOFTWARE\Cisco Systems, Inc.\Web Communicator\jsdkhostwinrect
Mac: jsdkhostwinrect@com.cisco.CiscoWebCommunicator

These values are deleted when the CiscoWebCommunicator Add-On is uninstalled.

Certificate Validation Feature Overview

Certificate Validation provides application developers with the ability to validate certificates used during user authentication against the certificate store on the client computer. Certificate Validation is mandatory for the Voice and Video SDK version 4.0.0 and later.

Certificate validation is used for the HTTP call made to the TFTP server to authenticate the user. Any TFTP server used for this purpose should have a valid certificate installed on the client computer. If invalid certificate is detected, sign in procedure is stopped and `'invalidCertificate.cwic'` event is triggered. Developers should handle this event, which carries along a certificate information and reasons to be rejected. If property `allowUserToAccept`, attached to event, is set to true, certificate could be accepted using `respond(certFingerprint, acceptBool)` function, also attached to event, else, certificate cannot be accepted, and only certificate info can be presented to user.

```

$('#phonecontainer').bind('invalidCertificate.cwic', handleInvalidCert);

function handleInvalidCert(event) {
    var certInfo = event.info,
        respond = event.respond;

    $('#certdetailstable .identifier').text(certInfo.identifierToDisplay);
    $('#certdetailstable .subjectCN').text(certInfo.certSubjectCN);
    $('#certdetailstable .reference').text(certInfo.referenceId);
    $('#certdetailstable .reason').text(certInfo.invalidReasons.join(', '));

    if (certInfo.allowUserToAccept) {
        respond(certInfo.certFingerprint, true); // or bind to button click event

        // or respond(certInfo.certFingerprint, false); to reject
    }
}

```

Windows

The self-signed certificate of the authentication server, or the root certificate of the Certificate Authority that signed the authentication server certificate, must be installed in the Current User or Machine account Trusted Root Certificate Authorities store. The Cisco Jabber Web SDK performs revocation list checking. If you are using a certificate authority, and the Certificate Authority policy requires CRL checking, it must be correctly configured in your network environment.

The Microsoft TechNet web site can provide more information about certificate management and usage in a Microsoft Windows environment. The Microsoft TechNet web site is available at <http://technet.microsoft.com>.

Mac OS X

The self-signed certificate of the authentication server, or the root certificate of the Certificate Authority that signed the authentication server certificate, must be installed in the user's Login or System store, the certificate must be marked Trusted for All Users, and marked as Always Trust for X.509 Basic Policy.

The Apple Support web site can provide more information about certificate management and usage in a Mac OS X environment. The Apple Support web site is available at <http://www.apple.com/support/>.

Service Discovery and Sign-in Types

From the release 11.0 of Cisco Jabber Web SDK, automatic discovery of users' home cluster is enabled. Service Discovery is a CUCM feature, adding to CUCM 9, which uses UDS API's to determine a users' home cluster from a central server running UDS and from there be able to login and download the appropriate information required for connecting to desired telephony device.

A users' home cluster is discovered by querying a Central UDS server in a given deployment. To find a Central UDS server a DNS Service Record (SRV) called `_cisco-uds` is queried. Jabber SDK knows what SRV record to query based on user's domain, which is collected by:

1. Prompting user for an email address from which the domain is extracted and cached.
2. Automatically discovering User Principal Name (UPN) on Windows OS machines. If Jabber SDK on Windows platforms fails to discover the UPN it will prompt the user to enter their email address. Jabber SDK on Mac OS X will always prompt the user to enter their email address if it is not cached.

After domain is acquired, it is cached on the file system and reused between sessions. If user wants to sign in on CUCM on different domain, `resetData` API must be called first.

After discovering its home node, Jabber SDK needs to discover if SSO is enabled or not. It is internally done

by querying a specific CUCM's API. Based on the result of this query, Jabber SDK will proceed with SSO or credentials based sign in.

Having service discovery feature present, now two types of sign in are possible:

1. **Manual sign in** - server addresses (tftp, ccmcip, cti) must be submitted along with other standard sign in parameters.
2. **Discovery based sign in** – server addresses are discovered automatically.

In the case of manual sign in, all required input parameters are submitted in advance, the same way as in previous versions of Jabber SDK, whereas in discovery based sign in, required inputs are prompted and submitted through the sign in lifecycle and appropriate callbacks.

Note: To use Service Discovery feature, user's home cluster must be configured appropriately. Configuration of UC services is outside the scope of this document. For more information, please refer to the [UCM maintenance and operations guide](#).

Jabber SDK Discovery Based Sign-In Lifecycle

Discovery based sign in is a preferred sign-in method in Jabber Web SDK 11.0. It is exposed through the new **startDiscovery** API, which initiates service discovery and sign-in lifecycle. The set of callback functions are added to the **settings** object in **init** API, which are used as handlers for sign-in lifecycle events. Those callbacks must be implemented if **settings.serviceDiscovery** parameter is not set to false (true by default). Use lifecycle callbacks to generate appropriate UI in your web application. Important difference between **registerPhone** API and new **startDiscovery** API is in a way of providing required inputs. Required inputs in the case of manual sign in are provided upfront, while in discovery based sign in required inputs are prompted and submitted through the sign in lifecycle and appropriate callbacks.

Sample sign-in flow based on service discovery

Before lifecycle events are described, let's see one of the possible sign-in flows:

1. User opens the web application with Jabber Web SDK for the first time and initiates discovery based sign-in
2. **emailRequired** callback is called, UI for entering email is generated and presented to user. User submits its email. If for some reason service discovery fails, error callback will be triggered and sign in attempt will be terminated.
3. Service discovery procedure has discovered SSO enabled user's home cluster after which **navigateTo.cwic** event is triggered with URL to navigate to from the popup window. After navigating to given URL, popup is redirected to authentication page where user authenticates with Identity Provider (IdP) which presents its own UI in opened popup. After successful authentication, token is acquired and popup window is closed. If SSO is not enabled on the home cluster, **credentialsRequired** callback would be called and user would submit credentials in the web app's UI instead of IdP's authentication UI.
4. **signedIn** callback is called. This callback signals that user is successfully authenticated with UCM, but still not connected to telephony device.
5. **devicesAvailable** callback is called.
6. User selects telephony device or it is automatically selected.
7. **args.success** callback is called when the selected device is successfully connected to UCM.

Lifecycle callbacks

Sign-in lifecycle callbacks are:

- emailRequired
- credentialsRequired
- signedIn

emailRequired callback

Handler for "Email Required" lifecycle event where user should provide its' email address, required by service discovery procedure.

This callback is called on:

- first use of Jabber SDK,
- after **resetData** API call,

To leave this state, **setEmail** function, passed as a parameter, should be called.

After the first use, Jabber SDK caches user email address and this callback is called again only in the case of service discovery error.

emailRequired callback signature is emailRequired(setEmail, cachedEmail):

1. setEmail(email) – function used to submit user's email address
2. cachedEmail – string value, if exists, it could be used to pre-populate input field in UI

credentialsRequired callback

Handler for "Credentials Required" lifecycle event where user should provide its' credentials.

This callback is called if non SSO credentials are not set

To leave this state, **setCredentials** function, passed-in as a parameter, should be called.

Note: Manual sign in is enabled through backward compatible **registerPhone** API, **credentialsRequired** callback should not be triggered in that case.

credentialsRequired callback signature is credentialsRequired(setEmail, cachedEmail):

1. setCredentials(username, passphrase) – function used to submit user's credentials,
2. cachedUser – string value, if exists, it could be used to pre-populate input field in UI

signedIn Callback

Handler for "Signed In" lifecycle event, meaning that Jabber SDK is successfully authenticated with desired CUCM. This state still does not indicate that application is connected to telephony device and it's ready to use. That state is indicated by the success callback, the same way as in previous Jabber SDK versions.

Shortly after **signedIn** callback, **devicesAvailable** callback should be called if set and desired device could be selected from provided list of available telephony devices.

Implementation of **signedIn** callback is optional. It could be used for adapting UI if such use case occurs.

signedIn callback signature is signedIn()

resetData API

After submitting email address in **emailRequired** callback, Jabber Web SDK add-on will cache the given value in the local storage. Any further discovery based sign in will use the cached value without emitting emailRequired event again. Having this behavior in mind, if connecting to another UCM is desired, **resetData** API must be called first. After calling **resetData**, **emailRequired** event will be emitted again after calling **startDiscovery** API.

Note: In EAP 1 Jabber Web SDK version, when changing sign in modes (from discovery based to manual or vice versa), **resetData** API must be called first. This behavior might be changed in the following releases.

Single Sign On

Single Sign On (SSO) simplifies the login process and offers easier and consistent way to authenticate across different services. Instead of each service requiring a separate username and password, a single credential allows access to all enabled services.

This document only covers how to use SSO with Cisco Unified Communication; it does not cover SSO fundamentals or concepts. The Cisco implementation of SSO is straightforward and easy to quickly setup and use without you having to understand the fundamentals of SSO. However, if you would like to learn more about the fundamentals of SSO, we recommend that you research the following topics:

- Security Assertion Markup Language (SAML)
- Open Authentication (OAuth)
- Identity Provider (IdP)
- Assertion Consumer Service (ACS)

To use SSO feature, user's home cluster must be SSO enabled first. Configuration of UC services for SSO is outside the scope of this document. For more information, please refer to the [UCM maintenance and operations guide](#).

Note: Jabber SDK SSO feature is supported with Cisco Unified Communications Manager v10.5+.

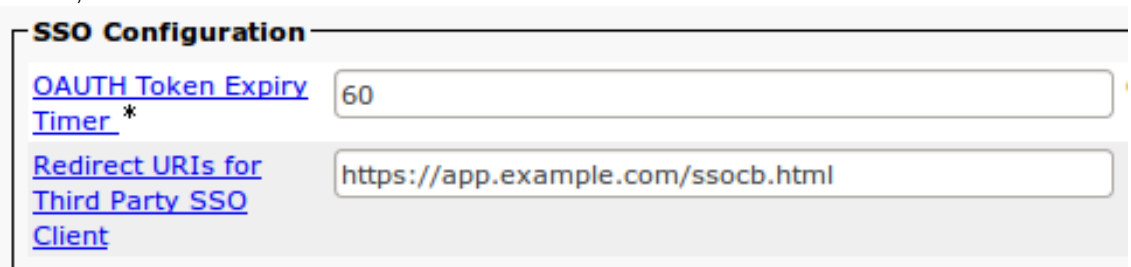
Understanding Redirects and Same-Origin Issues

For security reasons, web browsers prevent Javascript served by one origin from accessing data served by another origin (an origin is a combination of protocol, hostname, and port.) For example, Javascript on a page served from <https://app.example.com> cannot access the DOM of a popup window served from <https://ucmanager.example.com>, e.g. to retrieve a token.

The complete UC service SAML/OAuth SSO request sequence normally involves several HTTP interactions (302/Found redirects, Basic Auth challenge/response, scripted POST of a hidden HTML form, etc.) which involve requests served from different 'origins': i.e. the UC server and the Identity Provider. By default, the final HTTP location (and token info) is served from the UC service host (this normally prevents Javascript access to the token from other browser windows).

In order to facilitate this process, UCM allow the administrator to configure one or more 3rd party 'redirect URLs' to which the SSO sequence can be redirected after successful sign in. The application provides this redirect URL as a parameter when it initiates the SSO sign in sequence. The redirect URL should point to a web page hosted on the application web server, which will typically include some Javascript which extracts the SSO 'access_token' data and provides it back to a parent browser window.

The application redirect URL is configured in UCM in the Enterprise Parameters area (System->Enterprise Parameters->SSO Configuration), via the Redirect URIs for Third Party SSO Client parameter (multiple comma-separated URLs are allowed):



SSO Configuration

[OAUTH Token Expiry Timer *](#)

[Redirect URIs for Third Party SSO Client](#)

Setup Jabber SDK to use SSO feature

Jabber SDK's Single Sign On feature exist in the broader context of service discovery and sign-in lifecycle, as described in previous chapters. Jabber SDK SSO feature details will be explained in this chapter.

Before being able to sign in through SSO mode, developers/administrators should:

- Define discovery and sign-in lifecycle callbacks in Init API settings
- Configure redirect URI parameter on UCM (see previous chapter)
- Set redirectUri property on settings object (init API), to match the value configured on UCM
- Implement HTML file with script used for parsing token information. Save it to defined “redirect URI” location on application server.

Sample code for configuring settings object in init API:

```
var settings = {
  ready: jabberSDKReady,
  error: phoneErrorCallback,
  serviceDiscovery: true, // optional, defaults to true
  redirectUri: 'http://localhost:8000/samples/ssopopup.html',
  emailRequired: function handleEmailRequired(error, setEmail, cachedEmail) {
    if (error) {
      showError(error);
      return;
    }
    setEmail('user@mydomain.com');
  },
  credentialsRequired: handleCredentialsRequired,
  SSOSignInFailed: handleSSOSignInFailed,
  signedIn: handleSSOSignedIn
};
```

Sample HTML file containing the script for parsing token information:

```
<!doctype html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Parsing token on redirect URI</title>
</head>
<body>

<p>Sending token to parent page...</p>

<script>
  var href = document.location.href;

  window.opener._cwic_onSSOTokenReceived(href);
  window.close();

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

Single Sign On procedure begins by calling **startDiscovery** API. When service discovery is completed successfully and SSO enabled home cluster is found, Jabber SDK emits '**navigateTo.cwic**' public event. This event carries the information about the URL on which popup window or iFrame should be navigated to. When popup/iFrame window is navigated to given URL, it should be redirected to Identity Provider's authentication page where user should submit credentials. After user is successfully authenticated, popup/iFrame window is redirected to predefined 'redirect URI' with token information attached. Script at "redirect uri" location, should parse token data and pass it back to Jabber SDK, by calling special global function **_cwic_onSSOTokenReceived**. Further sign in flow is the same as for manual sign in, **devicesAvailable** callback is called, telephony device is selected and **success** callback is called after the device registration is successfully finished, or **error** callback if it fails.

Sample handler for **ssoNavigateTo.cwic** event:

```
function handleNavigateTo(event) {
    var url = event.url;

    //Open a child window for user interaction
    var popupForSSO = window.open('', '', 'height=500,width=500,scrollbars=1');
    popupForSSO.document.write('Authorizing...');
    //Start the SSO sequence using the request URL
    popupForSSO.location = url;
}
```

Note: When user is successfully authenticated with IdP, IdP creates a cookie in the current browser to keep the user signed in across different services and sessions, so user is not prompted for credentials again during the lifetime of the cookie. Type and the lifetime of created cookie is dependent on IdP configuration, i.e. it could be configured to create session or permanent cookies. It is important to note that during the lifetime of created cookie only the user for which the cookie is created can sign in with SSO. Even after calling **resetData** API, cookie will still be persisted in browser cookie store. This is in line with other web services supporting SSO. If you want to force the IdP to show the authentication UI again, cookie must be manually deleted from the cookie store.

Device Selection

The Jabber Web SDK provides means to choose from all available cameras, microphones, speakers, and other audio devices. This feature is available for the Voice and Video SDK version 4.0.0 and later.

In 11.0 release playout and ringer devices are separated, which means that different speaker devices could be used for conversation and ring. On Windows platforms, it is possible to set all available ringer devices as current ringer.

The device selection API consists of the following event and functions:

- **multimediaCapabilities.cwic event** – This event will be sent after multimedia capabilities are available in the add-on. This will be possible only after authentication.
- **mmDeviceChange.cwic event** – This event will be emitted whenever a change to the list of devices is detected. The application should use **getMultimediaDevices** API to acquire the updated list of devices.
- **getMultimediaDevices** – A function that retrieves a list of objects describing the multimedia devices installed on system.
- **setRecordingDevice** – A function that sets the audio recording device used by the add-on
- **setCaptureDevice** – A function that sets the video capture device used by the add-on
- **setPlayoutDevice** – A function that sets the play-out device used by the add-on
- **setRingerDevice** – A function that sets the ringer device used by the add-on
- **setPlayRingerOnAllDevices** – A function that sets all available ringer devices as current ringer device used by the add-on. (on Windows only)

Volume control on Windows platforms

In 11.0 release, Jabber Web SDK provides possibility of multimedia devices volume control. It is possible to set different volume levels on playout, ringer and recording devices. This feature is available on Windows platform only. The volume control API consists of the following events and functions:

- **setSpeakerVolume** – A function that sets the speaker volume to given value (0-100)
- **setRingerVolume** – A function that sets the ringer volume to given value (0-100)
- **setMicrophoneVolume** – A function that sets the microphone volume to given value (0-100)
- **getMultimediaDeviceVolume** – A function that returns volume level of the chosen device

Custom ringtones on Windows platforms

In 11.0 release, Jabber Web SDK introduces Custom ringtones feature, which brings possibility of using custom .wav files as ringtones. SDK comes with twelve preset different ringtones, but it is possible to expand that set with user-chosen files. Those files should be put in `installation_directory/Sound` directory.

This feature is available on Windows only.

The custom ringtones API consists of the following events and functions:

- **ringtonesListAvailable.cwic** – This event will be sent when list of ringtones is available.
- **ringtoneChange.cwic** – This event will be sent when current ringtone is changed.
- **setRingtone()** – A function that sets current ringtone to given value.

Node Component

The Cisco Jabber Voice and Video SDK node component—referred to as *node.csf*—is an optional, server-side component that brings the following advanced features to clients:

- Cisco Unified Communications Manager end-user authentication
- Phone device configuration retrieval
- Directory integration

node.csf exposes an HTTP REST API. The reference implementation is based on Node.js. *node.csf* is made up of independent modules. The reference implementation contains three modules:

- Hello:
 - Replies hello to clients
 - Used mainly for testing
- PhoneConfig:
 - Authenticates Cisco Unified Communications Manager end users
 - Retrieves list of users' phone devices
 - Downloads phone device configuration
- QuickContact—integrates with a directory to search for contacts

Install the Node Component

node.csf is based on the stable node.js version 0.8.x or later. node.csf currently works on Unix-based operating systems. It was tested on Linux and Mac OS X.

Procedure

- Step 1** Install the latest stable node.js version on a Unix-based operating system.
Tip: Some Linux distributions have binary packages (**nodejs**) or the source code can be downloaded and built directly. Go to <http://nodejs.org/#download>
- Step 2** To test node.js, run the command: `node -v`.
The version of **node.js** appears:
- ```
node -v
v0.10.5
```
- Step 3** node.js has a package manager called node package manager (npm). node.csf is provided as a npm package. To install npm, go to <https://www.npmjs.org/>.
- Step 4** To test that npm is installed successfully, run the command: `npm -v`.  
The version of npm appears:
- ```
npm -v
1.0.105
```
- Step 5** Installing the csf package requires the computer to be connected to the Internet temporarily so that extra dependencies can be obtained from the public npm repository. The **csf-x.y.x-rev.tgz** file is included in **CiscoJabberSDK.zip**. Download the zip file from the Jabber Web SDK Download and Docs page.
- Step 6** The installation builds a native component automatically, which requires the gcc compiler to be available. To test whether the gcc compiler is installed, run the command: `gcc -v`.
- Step 7** If the gcc compiler is not installed, refer to the OS documentation for information about installing compilers.
Tip: Some Linux distributions have build packages ready to be installed, such as the build-essential package on Ubuntu
- Step 8** Install the csf package by running one of the followings commands:
`$ npm install csf-3.0.0-61003.tgz`
If the machine is behind a proxy, use the `--proxy` option, for example:
`$ npm --proxy http://myproxy.com:8080 install csf-3.0.0-61003.tgz`
- Step 9** To test whether node.csf is installed, do one of the following:
1. Run the command: `$ node node_modules/csf/csf.js`.
 2. Open `http://localhost:1789/hello` in a web browser. node.csf replies.
Use a command-line tool such as cURL:
`$ curl -X GET http://localhost:1789/hello`
`$ curl -X POST http://localhost:1789/hello -d "test"`
This causes a server to start listening on port 1789 (default).
-

Node Component Configuration

node.csf can be configured using a text file that contains JSON. The main options are:

- Listening port (http or https)
- The execution environment:
 - *production* (recommended)
 - *beta* (dump exceptions with stack trace)
 - *development* (same as beta but with more logs)
- Some directories to be served statically (array of relative or absolute paths)

Cross-domain access Cross Origin Resource Sharing (CORS) is enabled by adding a corsproperty to the configuration file. The value is a JSON object passed to the connect-xcors module (empty object for default options).

node.csf also supports JSONP (JSON with padding), which provides another way of accessing cross-domain. This can be used with jQuery.ajax, for example, setting dataType to jsonp.

node.csf replies to a JSONP request by sending an object which can have either an errorproperty indicating a failure, or a dataproperty containing the requested data.

Secure http (HTTPS) can be enabled by adding a httpsproperty to the configuration file. The value is a JSON object with three properties: enabled(true or false), key, and certificate. A sample key and self-signed certificates are provided under the configdirectory. For more information, go to http://nodejs.org/api/tls.html#tls_tls_ssl.

node.csf loads applicative modules specified in the modulesmap of the configuration file. Each module is configured using either a JSON object, which can be empty, or a string referring to an extra configuration file specific to the module (absolute path or relative to the main configuration file path).

The template of the configuration file for node.csf is included in the package, under node_modules/csf/config/csf.cfg.

```
{
  "port": 1789,
  "env": "production",
  "staticDirs": [],
  "cors": {},
  "https": {
    "enabled": false,
    "key": "node_modules/csf/config/csf-key.pem",
    "cert": "node_modules/csf/config/csf-cert-self.pem"
  },
  "modules": {
    "hello": {},
    "phoneconfig": { /*For more information, see Node Component Modules.*/ },
    "quickcontact": { /*For more information, see Node Component Modules.*/ }
  }
}
```

Related Topics

[Node Component Modules, on page 37](#)

Run the Node Component

Procedure

- Step 1** To start node.csf with the default configuration, change to the installation directory and run the following command:
- ```
$ node node_modules/csf/csf.js
```
- Step 2** To specify a configuration, enter a command similar to the following:
- ```
$ node node_modules/csf/csf.js myconfigfile.cfg
```
- Step 3** To run node.csf in the background, use the **nohup** command. For example:
- ```
$ nohup node node_modules/csf/csf.js myconfigfile.cfg > node.out 2> node.err < /dev/null &
```
- Tip:** You can also use the Forever utility. For more information, go to <http://blog.nodejitsu.com/keep-a-nodejs-server-up-with-forever/>

# Node Component Modules

The following tables:

- Describe the modules in the node component.
- Provide examples of the configuration file for the node component modules.

**Table 1: Node Component Modules**

| Module      | Description                                                                                                                                                                                                                    | API Exposed                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Hello       | <ul style="list-style-type: none"> <li>• Used mainly for testing.</li> <li>• Replies <b>hello</b> to any received request.</li> <li>• The module is always loaded, even if not listed in the configuration modules.</li> </ul> | <p><b>GET /hello</b></p> <p>Replies Hello with the current date and time</p> <p><b>POST /hello</b></p> <p>Replies with <b>Hello {post data}</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| PhoneConfig | Interacts with Cisco Unified Communications Manager to provide phone configuration to the Cisco Jabber SDK                                                                                                                     | <p><b>GET /phoneconfig/devices?ccmcip={ccmcip}</b></p> <ul style="list-style-type: none"> <li>• Authenticates (HTTP Basic) end user against Cisco Unified Communications Manager with address {ccmcip}</li> </ul> <p><b>Important</b> If your Cisco Unified Communications Manager server uses a self-signed certificate, to use this module you must add the <b>rejectUnauthorized</b> parameter to the configuration file. This parameter allows connections to servers with untrusted certificates.</p> <ul style="list-style-type: none"> <li>• Returns a JSON array with all phone devices associated to the user. Each device is a JSON object with the properties: <ul style="list-style-type: none"> <li>• name: device name (String)</li> <li>• model: device model (String)</li> <li>• description: device description (String)</li> </ul> </li> </ul> <p><b>GET /phoneconfig/device/user/{user}/client/cwc?tftp={tftp}</b></p> <p>Returns the phone device configuration (XML) associated to user {user} and client type <b>cwc</b>. Configuration is a JSON object with the properties:</p> <ul style="list-style-type: none"> <li>• name: device name (String)</li> <li>• tftp: the successful TFTP address (String)</li> <li>• configuration: a string containing the XML <b>&lt;device&gt;...&lt;/device&gt;</b></li> </ul> |

| Module       | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | API Exposed                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QuickContact | <ul style="list-style-type: none"> <li>Integrates with a directory to provide contact information to clients.</li> <li>Currently supports LDAP to search for a directory.</li> <li>Uses ldaps to connect to the directory For more information, go to <a href="http://ldaps.org/">http://ldaps.org/</a>.</li> </ul> <p>The QuickContact module must be configured with:</p> <ul style="list-style-type: none"> <li>The LDAP server address</li> <li>The LDAP principal credentials (read only is acceptable)</li> <li>The search tree base</li> <li>The attributes mapping</li> </ul> <p>The <b>photosrc</b> property can contain the <i>{screenName}</i> pattern to be replaced with the screenNamevalue, for example: "photosrc":<br/> "http://mydirectory.com/photo/{screenName}.jpg"</p> | <p><b>GET</b><br/> /<i>quickcontact/name/{name}</i>?max={max}&amp;photo={photo}</p> <ul style="list-style-type: none"> <li>Searches contacts whose screen, first, last, and display name starts with {name}.</li> <li>{max} is the maximum number of contacts to return. 0 means unlimited (default), absolute max is 20.</li> <li>If {photo} equals yes, each result includes a photosrc property.</li> <li>Returns an array (possibly empty) of contact objects.</li> </ul> <p><b>GET</b><br/> /<i>quickcontact/email/{email}</i>?max={max}&amp;photo={photo}</p> <ul style="list-style-type: none"> <li>Searches contacts whose email address starts with {email}.</li> <li>{max} is the maximum number of contacts to return. 0 means unlimited (default), absolute max is 20.</li> <li>If {photo} equals yes, each result includes a <b>photosrc</b> property.</li> <li>Returns an array (possibly empty) of contact objects.</li> </ul> <p><b>GET</b><br/> /<i>quickcontact/phoneNumber/{phoneNumber}</i>?cc={cc}&amp;max={max}&amp;photo={photo}</p> <ul style="list-style-type: none"> <li>Searches contacts whose phone number starts with {phoneNumber} (sequence of digits).</li> <li>If {cc} equals yes, {phoneNumber} starts with a country code (false by default).</li> <li>{max} is the maximum number of contacts to return. 0 means unlimited (default), absolute max is 20.</li> <li>If {photo} equals yes, each result includes a photosrc property.</li> <li>Returns an array (possibly empty) of contact objects.</li> <li>{phoneNumber} should contain as many digits as possible to narrow the search down to fewer contacts.</li> </ul> <p><b>Note</b>      Search by phone number is not fully optimized yet and can result in large requests on the directory (LDAP) server.</p> |

Table 2: Node Component Modules Configuration File Examples

| Module      | Configuration file example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| PhoneConfig | <pre>"phoneconfig": {   "cucmCluster": {     "firstCluster": {       "primaryNode": "192.168.1.1",       "nodes": ["192.168.1.1", "192.168.1.2"],       "tftpNodes": ["192.168.1.1"]     }   },   "cucmClusterDnRule": [     ["1234XXX", "firstCluster"],     ["+12345678XXX", "firstCluster"]   ],   "CertificateAuthorization": {     "rejectUnauthorized": false   } },</pre> <p>Example:</p> <pre>GET /phoneconfig/cucmCluster/name/firstCluster {   "primaryNode": "192.168.1.1",   "nodes": ["192.168.1.1", "192.168.1.2"],   "tftpNodes": "192.168.1.1" }</pre> |

| Module       | Configuration file example                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| QuickContact | <pre> "quickcontact": {   "ldap": {     "host": "192.168.1.1",     "port": 1234,     "credentials": {       "user": "CN=cn,OU=ou,DC=dc,DC=dc2",       "password": "12345"     },     "treebase": "OU=ou,OU=ou2,DC=dc,DC=dc2",     "mapping": {       "screenName": "screenName",       "displayName": "display",       "email": "mail",       "phoneNumbers": {         "work": "workNumber",         "mobile": "mobileNumber"       },       "photosrc": "http://www.company.com/dir/photos/{screenName}.jpg",     }   } } </pre> <p>Example:</p> <pre> GET /quickcontact/name/jsmith?photo=yes {   "screenName": "jsmith",   "displayName": "John Smith",   "email": "jsmith@company.com",   "phoneNumbers": {     "work": "+1 234 56 78",     "mobile": "+9 876 54 32"   },   "photosrc": "http://www.company.com/dir/photo/jsmith.jpg" } </pre> |



# Directory Navigator Sample Application

The Directory Navigator is a single-page sample application that allows users to search a user directory from a web page. If your application is using the Cisco Jabber SDK node component for directory integration, the Directory Navigator allows you to add search and retrieval of directory profiles. The sample application illustrates the use of the node REST API, and can be used as an example to build on and incorporate into your web application.

The Directory Navigator:

- Is built on the Cisco Jabber SDK node component.
- Uses the quickcontact node module to search the directory and retrieve profiles.
- Can be used with the Chrome, Firefox, and Safari browsers. There are known display issues when the application is used with Internet Explorer browser.

The Directory Navigator user interface uses the Twitter Bootstrap library. The interface is fully responsive and adapts to different screen sizes for smartphones, tablets, and desktops.

Users search the directory by entering a name in the search bar at the top of the application. As shown in the following figure, for each contact that matches the search criteria, the following information appears:

- Contact photo
- Display name
- Number of direct reports (in parentheses next the display name)
- Job title
- Location

**Figure 1: Directory Navigator Search Results**





When users click one of the search results, the contact profile appears, as shown in the following figure.

Figure 2: Directory Navigator Contact Profile

Fox

Charles Holland





**Nancy Fox** 


Manager - Software Engineering  
Newark, California, United States

Email Chat Call ▾

3 reports

 **Anita Perez**  
Engineer - Software Engineering  
Newark, California, United States

 **Neela Patel**  
Engineer - Software Engineering  
Newark, California, United States

 **Sue Miller**  
Developer - Software Engineering  
Newark, California, United States

372367

The contact profile includes the following additional information:

- Links to up to two levels of managers (the names of the managers appear above the contact photo)
- Buttons that allow users to:
  - Send an email.
  - Initiate a Cisco Jabber chat or call. Cisco Jabber must be installed locally.
- Links to the direct reports of the contact

## Set Up Directory Navigator

The Directory Navigator queries the node quickcontact REST API to search contacts by name and retrieve contact information. If the `serveSamplesconfiguration` parameter is set to true, the Directory Navigator is served automatically by the SDK node at `/quickcontact/samples/dirnavigator.html`.

```
{
 ...
 "modules": {
 "quickcontact": {
 "serveSamples": true,
 ...
 }
 }
}
```

The Directory Navigator expects the following contact keys mapped to LDAP attributes in the SDK node configuration file:

```
"mapping": {
 "screenName": "<ldap>",
 "displayName": "<ldap>",
 "email": "<ldap>",
 "phoneNumbers": {
 "work": "<ldap>",
 "mobile": "<ldap>"
 },
 "photosrc": "http://mydir.com/photos/{screenName}.jpg",
 "title": "<ldap>",
 "city": "<ldap>",
 "state": "<ldap>",
 "country": "<ldap>",
 "distinguishedName": "<ldap>",
 "manager": "<ldap>",
 "directReports": "<ldap>"
}
```



### Note

- The `photosrc` attribute is an image source URL used for contact photos. The URL can contain a `{screenName}` token expanded to the actual screen name. The URL should be reachable from clients (browsers).
- The `phoneNumbers` value must contain an object whose keys are phone type and values are phone numbers. The application capitalizes phone types.
- The `distinguishedName` and `manager` values must contain an LDAP fully qualified distinguished name (FQDN).
- The `directReports` attribute must contain an array of FQDNs.

Contact result example:

```
"mapping": {
 "screenName": "nfox",
 "displayName": "Nancy Fox",
 "email": "nfox@example.com",
 "phoneNumbers": {
 "work": "+1 408 555 1514",
 "mobile": "+1 408 555 1515"
 },
 "photosrc": "http://mydir.com/photos/nfox.jpg",
 "title": "Manager - Software Engineering",
 "city": "Newark",
 "state": "CA",
 "country": "USA",
 "distinguishedName": "CN=nfox, DC=mycompany, DC=com",
 "manager": "CN=boss, DC=example, DC=com",
 "directReports": [
 "CN=aperez, DC=example, DC=com",
 "CN=npatel, DC=example, DC=com",
]
}
```



## CHAPTER 6

# Troubleshooting

---

- [Debugging JavaScript, page 44](#)
- [JavaScript Events for Microsoft Windows Video Window, page 45](#)
- [Disable App Nap, page 45](#)
- [Common Errors, page 45](#)
- [Uninstall Cisco Web Communicator Add-on, page 46](#)
- [Log Files, page 46](#)

## Debugging JavaScript

If you experience problems while using the Cisco Jabber SDK your first step is to open up the JavaScript console and other debugging tools in your browser.

### Chrome

Chrome includes a good set of tools. Click the **Spanner** icon and choose **Tools > JavaScript Console**.

### Firefox

Firefox includes a good set of tools. Choose **Tools > Web Developer > Web Console**.

### Internet Explorer

Choose **Tools > Developer Tools** (or press F12). Click the **Script** tab to open the Console view in the right pane. The Console view displays any error messages from the Cisco Jabber SDK or other errors that may be causing problems.

### Safari

By default, the developer tools in Safari are hidden. To activate the tools, choose **Preferences > Advanced** and check the option **Show Develop menu in menu bar**. This activates the Develop menu which contains several tools. To display the console, select **Show Error Console**.

# JavaScript Events for Microsoft Windows Video Window

On Microsoft Windows, the video window object created by `cwic createVideoWindow` does not generate HTML mouse and click events. A workaround is to attach the video window element to an invisible `<div>` (`style="display: none"`) and capture events from that element. Also on Microsoft Windows, HTML elements like buttons will not display on top of the video windows; the video windows always render on top. These limitations do not apply to video windows on Mac.

## Disable App Nap

We recommend that App Nap, if the Mac OS X version being used has that feature, be disabled for all browsers supported by the Cisco Jabber Web SDK.

### Procedure

- Step 1** For Chrome and Firefox:
- In Finder, right-click the browser's .appfile and click **Get Info**.
  - Check the **Prevent App Nap** check box.
- Step 2** For Safari:
- Open Terminal.
  - Run the following command:  
`defaults write -app Safari NSAppSleepDisabled -bool YES`

## Common Errors

### TFTP fetch error

The most common reason for this error is that you have not set up a device in Cisco Unified Communications Manager. For instructions on setting up a device, see `JabberSDK_CUCM_Guide.txt` in the `docs` folder. Another possible cause for this error is a firewall that could be blocking TFTP access from your browser. For more information, check with your system administrator.

### Add-on not installed

Make sure that you have installed the browser add-on.



#### Note

If you have disabled the add-on, you may be prompted, in error, to install the add-on even though the add-on is already installed. Check to see if the add-on is disabled in your browser.

# Uninstall Cisco Web Communicator Add-on

## Procedure

Do one of the following:

| Option  | Description                                                                                                                                                                                                                                                                                                                                                                                                              |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Mac     | <p>Run the following commands from a terminal window:</p> <pre>sudo rm -rf /Library/Internet\ Plug-Ins/CiscoWebCommunicator.plugin/ sudo pkgutil --forget com.cisco.pkg.CiscoWebCommunicator</pre> <p>For Mac OS X 10.10, after running the previous commands, run the following commands:</p> <pre>sudo rm -rf ~/Library/Application\ Support/CiscoWebCommunicator defaults delete com.cisco.CiscoWebCommunicator</pre> |
| Windows | <ol style="list-style-type: none"><li>1 From the <b>Control Panel</b>, click <b>Program and Features</b>.</li><li>2 Click <b>Cisco Web Communicator</b>.</li><li>3 Click <b>Uninstall</b>.</li><li>4 Follow the instructions on the screen.</li></ol>                                                                                                                                                                    |

## Log Files

The add-on the SDK uses to enable telephony writes log files to your local file system. You can find the log files in the following locations:

- Windows  
C:\Users\{username}\AppData\Local\CiscoWebCommunicator\softphone.log
- Mac OS X:  
/Users/{username}/Library/Application Support/CiscoWebCommunicator/softphone.log

7:

While it is not really intended as a diagnostic tool for SDK users, softphone.log contains a wealth of detailed information and can be useful when you contact Cisco support.