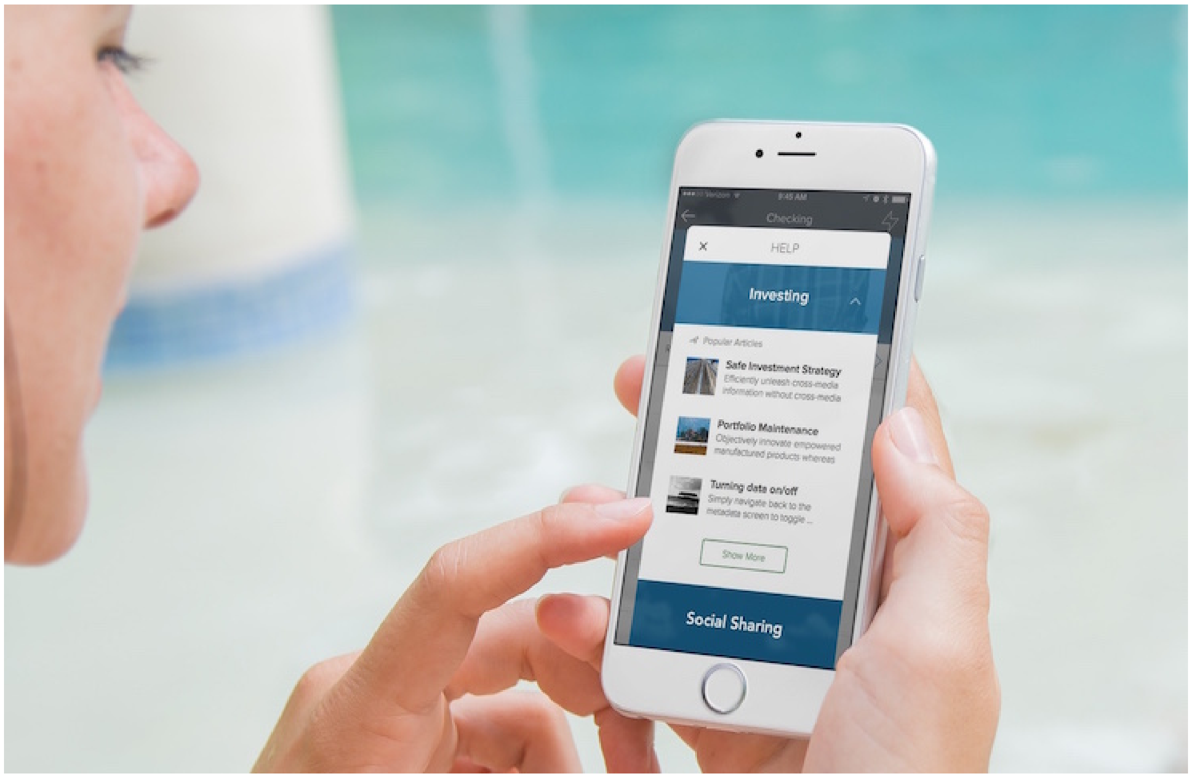
Embedded Service SDK for Android Developer Guide

Version 224.0



@salesforcedocs

Last updated: May 25, 2021

© Copyright 2000–2020 salesforce.com, inc. All rights reserved. Salesforce is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

CONTENTS

EMBEDDED SERVICE SDK FOR ANDROID DEVELOPER GUIDE

The Embedded Service SDK for Mobile Apps makes it easy to give customers access to powerful features right from within your native app. You can make these Service Cloud features feel organic to your app and have things up and running quickly using this SDK.

April 2021 Release (Version 224.1.0)

This documentation describes the Service SDK, which uses the following components.

Component	Version Number
Chat	4.2.0
Knowledge	4.3.4
Case Management	4.2.5
SOS	4.0.9
Common	8.0.2

Release Notes
Check out the new features and known issues for the Android Service SDK.

Service Cloud Setup for the Embedded Service SDK for Mobile Apps
Set up Service Cloud in your org before using the Service SDK.

Embedded Service SDK for Mobile Apps Setup
Set up the SDK to start using Service Cloud features in your mobile app.

Android Examples
Use these examples to learn more about the Service SDK.

Using Chat with the Service SDK
Add the Chat experience to your mobile app.

Using Knowledge with the Service SDK
Add the Knowledge experience to your mobile app.

Using Case Management with the Service SDK
Add the Case Management experience to your mobile app.

Using SOS with the Service SDK
Add the SOS experience to your mobile app.

SDK Customizations with the Service SDK for Android
Once you've played around with some of the SDK features, use this section to learn how to customize the Service SDK so that it fits the look and feel of your app. This section also contains instructions for localizing strings in all supported languages.

Troubleshooting the Service SDK
Get some guidance when you run into issues.

Data Protection and Security in the Service SDK for Android
The Service SDK does not collect or store personal data from its users. We ensure that data is secure both locally and when in transit.

Reference Documentation
Reference documentation for Service SDK for Android.

Additional Resources
If you're looking for other resources, check out this list of links to related documentation.

Release Notes

Check out the new features and known issues for the Android Service SDK.

To review the latest releases for the Service SDK for Android, visit github.com/forcedotcom/ServiceSDK-Android/releases.

Service Cloud Setup for the Embedded Service SDK for Mobile Apps

Set up Service Cloud in your org before using the Service SDK.

Org Setup for Chat in Lightning Experience with a Guided Flow
Use the guided setup flow in Lightning Experience to add chat to your org.

Org Setup for Chat in Salesforce Classic
To use Chat in your mobile app, first set up Chat in your org.

Cloud Setup for Knowledge
To use Knowledge in your mobile app, enable it in your org, create knowledge articles, and set up an Experience Cloud.

Cloud Setup for Case Management
To use Case Management in your mobile app, set up an Experience Cloud site and create a quick action.

Console Setup for SOS
To use SOS in your mobile app, set up Omni-Channel and SOS for your console.

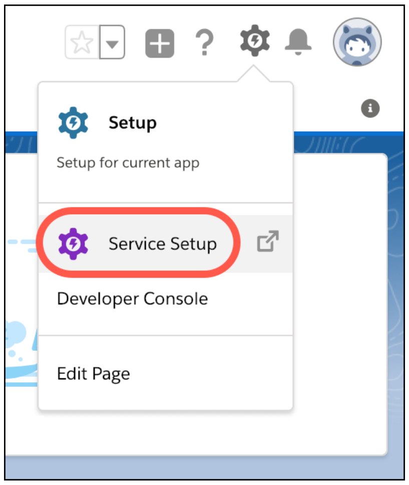
Org Setup for Chat in Lightning Experience with a Guided Flow

Use the guided setup flow in Lightning Experience to add chat to your org.

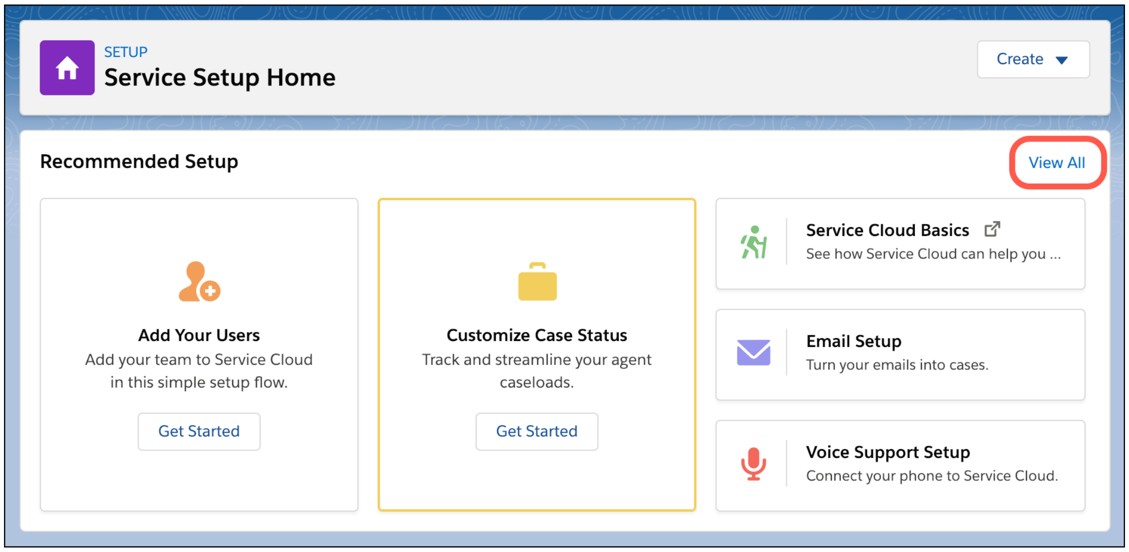
Note: If you're using Salesforce Classic, see Org Setup for Chat in Salesforce Classic.

These instructions walk you through a basic chat setup in Lightning Experience. To learn more about chat, check out the Web Chat Basics Trailhead module.

1. Click the Setup gear icon and select **Service Setup**.



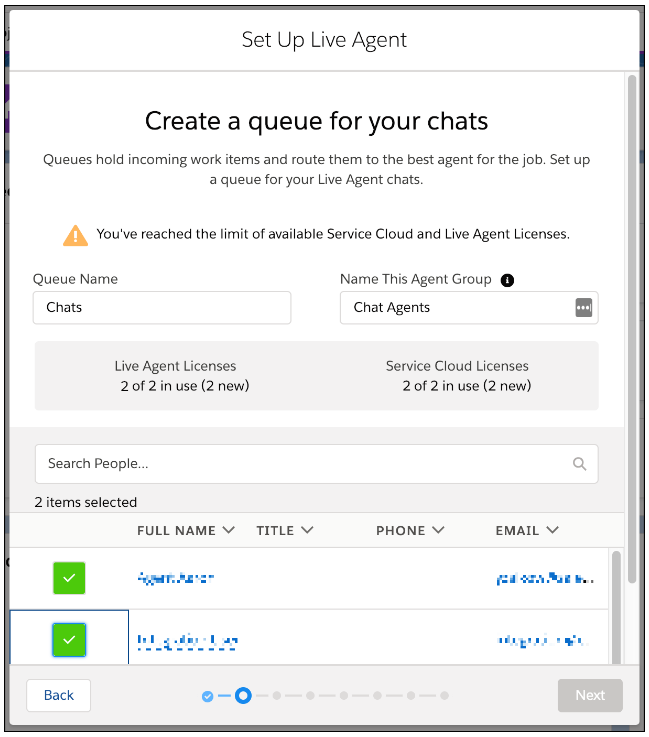
2. Under Recommended Setup, click **View All**.


SETUP
Service Setup Home
Create
Recommended Setup
View All
Add Your Users
Add your team to Service Cloud in this simple setup flow.
Get Started
Customize Case Status
Track and streamline your agent caseloads.
Get Started
Service Cloud Basics
See how Service Cloud can help you ...
Email Setup
Turn your emails into cases.
Voice Support Setup
Connect your phone to Service Cloud.

3. In the search box, enter `Chat`, and select **Chat with Customers**.

   Note: If you don't see the **Chat with Customers** setup flow, verify that your org includes the Digital Engagement add-on SKU.

4. After you read the overview page, click **Start**.
5. Enter the name of your queue (for example, `Chats`) and agent group name (for example, `Chat Agents`). Then select the members for this group and click **Next**.


Set Up Live Agent

Create a queue for your chats

Queues hold incoming work items and route them to the best agent for the job. Set up a queue for your Live Agent chats.

⚠ You've reached the limit of available Service Cloud and Live Agent Licenses.

Queue Name: Chats
Name This Agent Group: Chat Agents

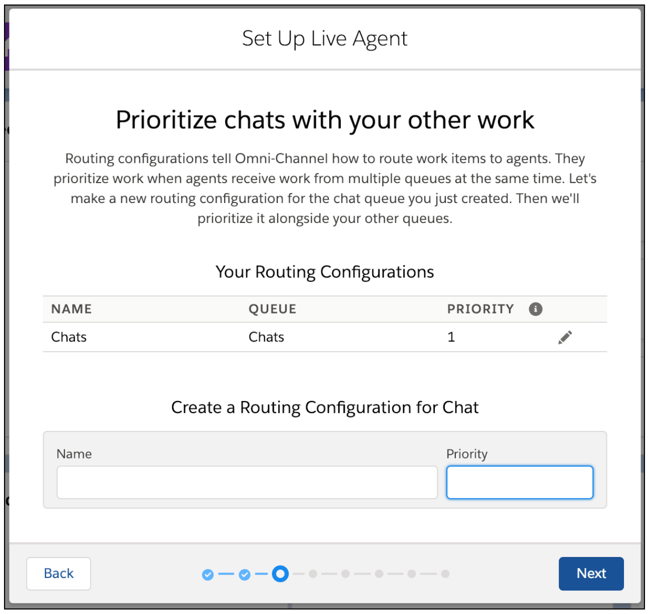
Live Agent Licenses 2 of 2 in use (2 new)
Service Cloud Licenses 2 of 2 in use (2 new)

Search People...

2 items selected

	FULL NAME	TITLE	PHONE	EMAIL
✓	[redacted]			[redacted]
✓	[redacted]			[redacted]

6. If you're asked to prioritize chats with your other work, enter the routing configuration name (for example, `Chats`) and give it a priority (for example, *1*).


Set Up Live Agent

Prioritize chats with your other work

Routing configurations tell Omni-Channel how to route work items to agents. They prioritize work when agents receive work from multiple queues at the same time. Let's make a new routing configuration for the chat queue you just created. Then we'll prioritize it alongside your other queues.

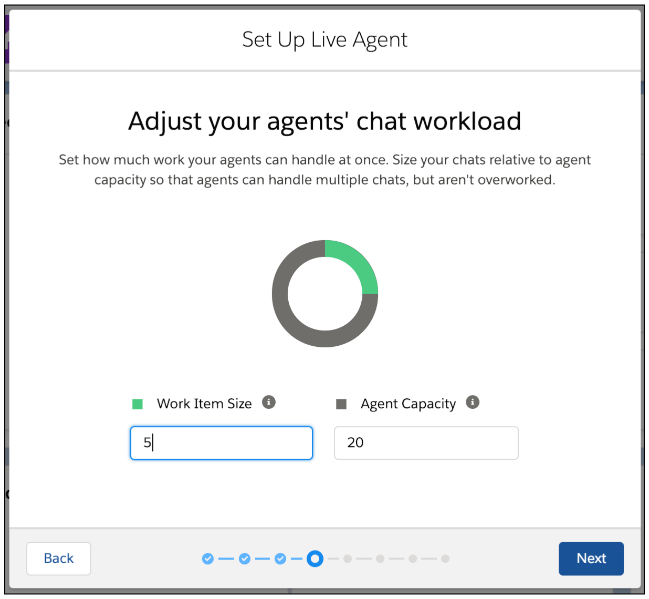
Your Routing Configurations

NAME	QUEUE	PRIORITY
Chats	Chats	1

Create a Routing Configuration for Chat

Name:
Priority:

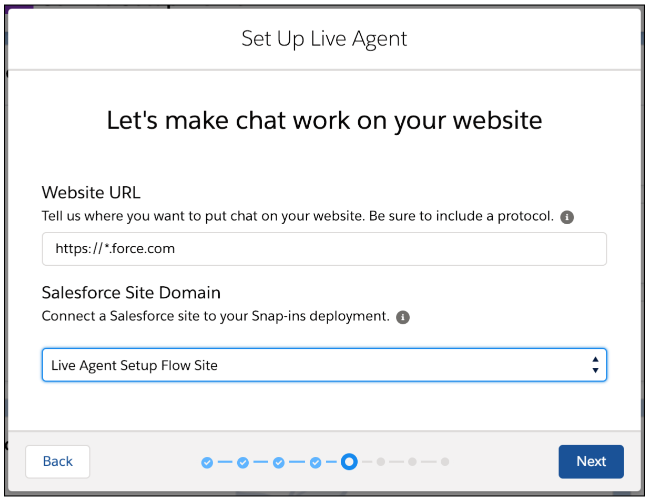
7. (Optional) Adjust the work item size and agent capacity.


Set Up Live Agent

Adjust your agents' chat workload

Set how much work your agents can handle at once. Size your chats relative to agent capacity so that agents can handle multiple chats, but aren't overworked.

8. For the website URL, enter `https://*.force.com` or the URL for your site. Create or select a Salesforce site.


Set Up Live Agent

Let's make chat work on your website

Website URL

Tell us where you want to put chat on your website. Be sure to include a protocol.

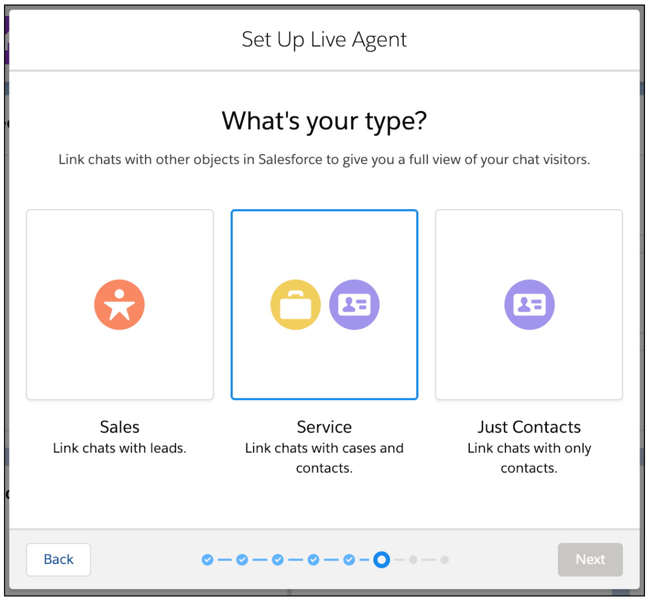
https://*.force.com

Salesforce Site Domain

Connect a Salesforce site to your Snap-ins deployment.

Live Agent Setup Flow Site

9. For the type of chat, select **Service**.



Set Up Live Agent

What's your type?

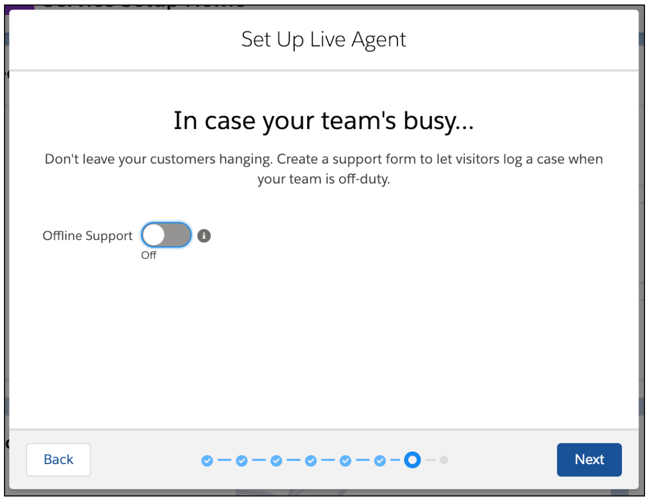
Link chats with other objects in Salesforce to give you a full view of your chat visitors.

Sales
Link chats with leads.

Service
Link chats with cases and contacts.

Just Contacts
Link chats with only contacts.

10. Choose whether you want to provide offline support for customers.



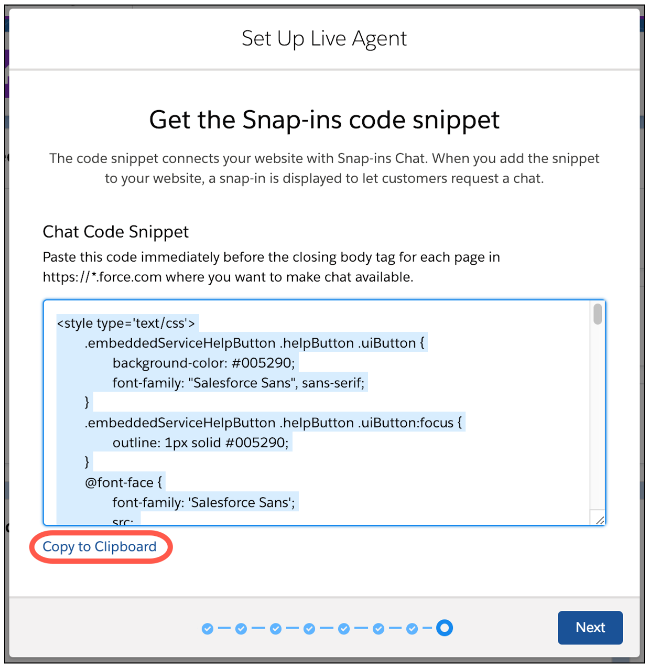
Set Up Live Agent

In case your team's busy...

Don't leave your customers hanging. Create a support form to let visitors log a case when your team is off-duty.

Offline Support Off

11. Copy the code snippet by clicking **Copy to Clipboard**, and paste it into a text editor. You must extract a few pieces of information from this code snippet.



Get the Snap-ins code snippet

The code snippet connects your website with Snap-ins Chat. When you add the snippet to your website, a snap-in is displayed to let customers request a chat.

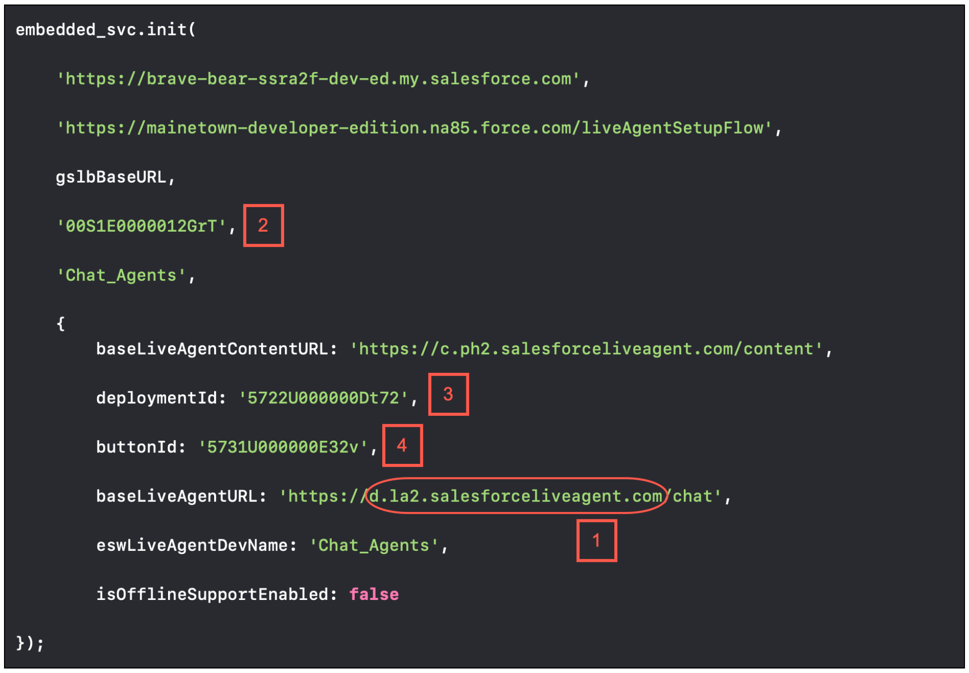
Chat Code Snippet

Paste this code immediately before the closing body tag for each page in https://*.force.com where you want to make chat available.

```
<style type='text/css'>
    .embeddedServiceHelpButton .helpButton .uiButton {
        background-color: #005290;
        font-family: "Salesforce Sans", sans-serif;
    }
    .embeddedServiceHelpButton .helpButton .uiButton:focus {
        outline: 1px solid #005290;
    }
    @font-face {
        font-family: 'Salesforce Sans';
        src:
```

12. In the text editor, copy the following configuration information from the `embedded_svc.init` function.

- (1) Chat Endpoint Hostname—This value is the hostname of the `baseLiveAgentURL` property. When copying the hostname, be sure not to include the protocol or the path. For instance, if the value for `baseLiveAgentURL` is `https://`***`MyDomainName`***`.my.salesforcescrt.com/chat`, then the hostname is ***`MyDomainName`***`.my.salesforcescrt.com`.
- (2) Org ID—If you don't already know this value, it's the fourth argument in the `embedded_svc.init` function call.
- (3) Deployment ID—This value can be found in the `deploymentId` property.
- (4) Button ID—This value can be found in the `buttonId` property.

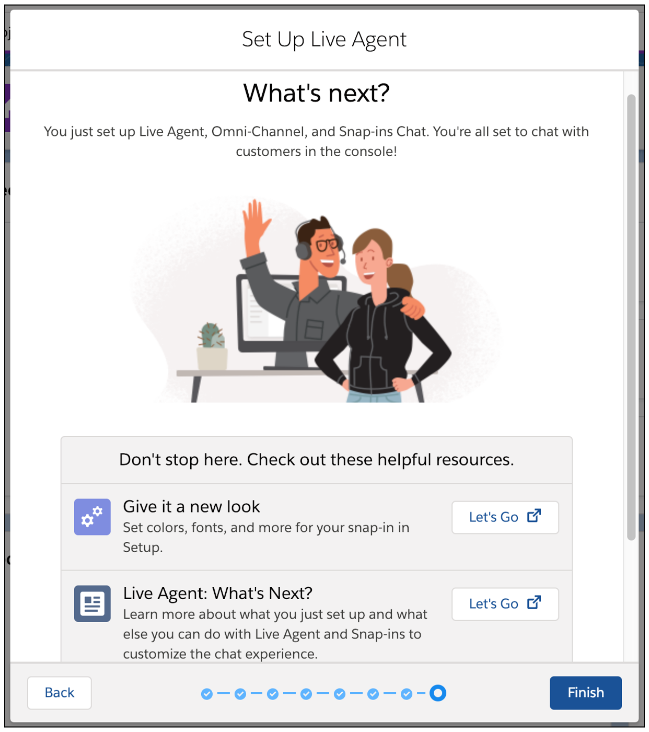


```
embedded_svc.init(
    'https://brave-bear-ssra2f-dev-ed.my.salesforce.com',
    'https://mainetown-developer-edition.na85.force.com/liveAgentSetupFlow',
    gslbBaseURL,
    '00S1E0000012GrT',
    'Chat_Agents',
    {
        baseLiveAgentContentURL: 'https://c.ph2.salesforceliveagent.com/content',
        deploymentId: '5722U000000Dt72',
        buttonId: '5731U000000E32v',
        baseLiveAgentURL: 'https://d.la2.salesforceliveagent.com/chat',
        eswLiveAgentDevName: 'Chat_Agents',
        isOfflineSupportEnabled: false
});
```

Give these four settings to your developer.

Note: If you don't copy this information now, you can copy it later using the instructions in Get Chat Settings from Your Org.

13. Go back to the guided setup flow and click **Finish**.



Set Up Live Agent

What's next?

You just set up Live Agent, Omni-Channel, and Snap-ins Chat. You're all set to chat with customers in the console!

Don't stop here. Check out these helpful resources.

Give it a new look
Set colors, fonts, and more for your snap-in in Setup.

Live Agent: What's Next?
Learn more about what you just set up and what else you can do with Live Agent and Snap-ins to customize the chat experience.

14. (Optional) If you want to build a chatbot to complement your chat experience, see Einstein Bots in Salesforce Help. In broad strokes, you must enable Einstein Bots, deploy the bot to your channel, and activate the bot. If you want to learn about building a more robust bot, see the Einstein Bots Developer Cookbook.

You're all set! Chat is now set up in your org. You can always fine-tune these settings from **Setup**. To learn more, see Chat in Salesforce Help.

Note: To learn about chat timeout limitations on iOS devices, see When does a chat session time out?

Get Chat Settings from Your Org

After you've set up chat in the console, supply your app developer with four values: the chat endpoint hostname, the organization ID, the deployment ID, and the button ID. You can get this information from your org's setup.

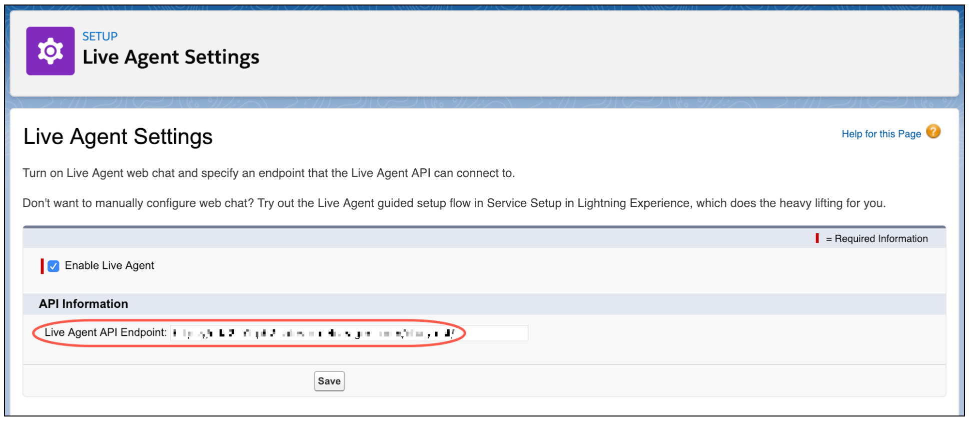
Get Chat Settings from Your Org

After you've set up chat in the console, supply your app developer with four values: the chat endpoint hostname, the organization ID, the deployment ID, and the button ID. You can get this information from your org's setup.

Note: If the endpoint for your server changes (due to an org migration, for example), the SDK automatically reroutes you to the correct server. However, to avoid unnecessary rerouting, you should still update the server endpoint when you notice it has changed inside your org's settings.

Chat Endpoint Hostname

The hostname for the Chat endpoint that your organization has been assigned. To get this value, from Setup, search for **Chat Settings** and copy the hostname from the **API Endpoint**.



SETUP

Live Agent Settings

Live Agent Settings

Help for this Page

Turn on Live Agent web chat and specify an endpoint that the Live Agent API can connect to.

Don't want to manually configure web chat? Try out the Live Agent guided setup flow in Service Setup in Lightning Experience, which does the heavy lifting for you.

= Required Information

✓ Enable Live Agent

API Information

Live Agent API Endpoint: [redacted]

Save

Be sure not to include the protocol or the path. For instance, if the API Endpoint is:

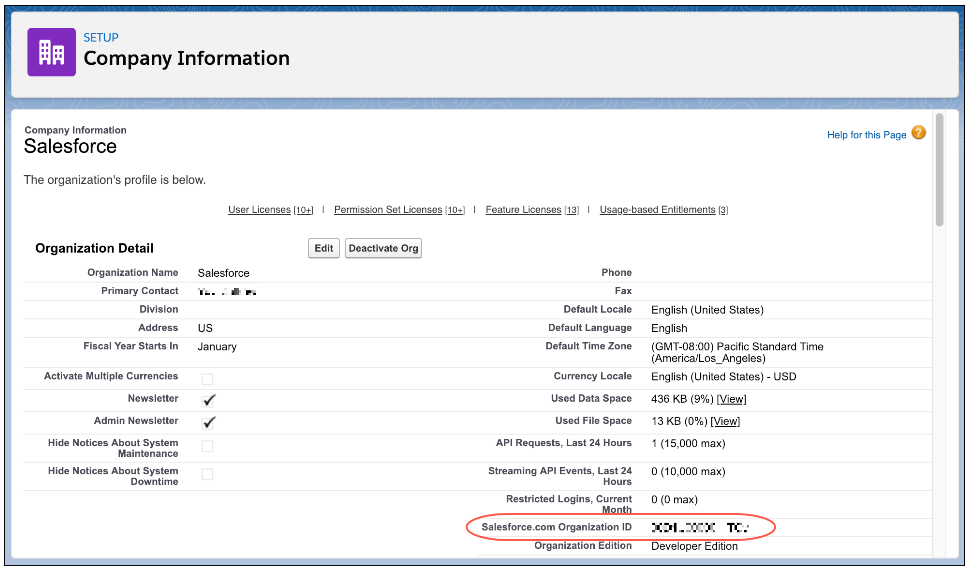
```
https://d.gla5.gus.salesforce.com/chat/rest/
```

The chat endpoint hostname is:

```
d.gla5.gus.salesforce.com
```

Org ID

The Salesforce org ID. To get this value, from Setup, search for **Company Information** and copy the **Salesforce Organization ID**.


SETUP
Company Information

Company Information
Salesforce

The organization's profile is below.

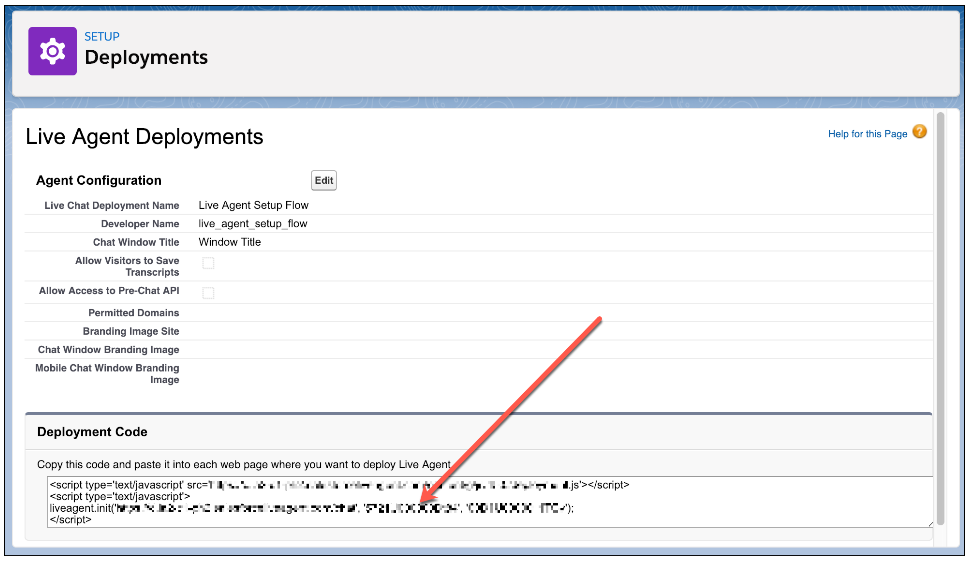
User Licenses [10+] | Permission Set Licenses [10+] | Feature Licenses [13] | Usage-based Entitlements [3]

Organization Detail

Organization Name	Salesforce	Phone	
Primary Contact	[redacted]	Fax	
Division		Default Locale	English (United States)
Address	US	Default Language	English
Fiscal Year Starts In	January	Default Time Zone	(GMT-08:00) Pacific Standard Time (America/Los_Angeles)
Activate Multiple Currencies	☐	Currency Locale	English (United States) - USD
Newsletter	✓	Used Data Space	436 KB (9%) [View]
Admin Newsletter	✓	Used File Space	13 KB (0%) [View]
Hide Notices About System Maintenance	☐	API Requests, Last 24 Hours	1 (15,000 max)
Hide Notices About System Downtime	☐	Streaming API Events, Last 24 Hours	0 (10,000 max)
		Restricted Logins, Current Month	0 (0 max)
		Salesforce.com Organization ID	[redacted]
		Organization Edition	Developer Edition

Deployment ID

The unique ID of your Chat deployment. To get this value, from Setup, select **Chat** > **Deployments**. The script at the bottom of the page contains a call to the `liveagent.init` function with the **pod**, the **deploymentId**, and **orgId** as arguments. Copy the **deploymentId** value.


SETUP
Deployments

Live Agent Deployments

Agent Configuration

Live Chat Deployment Name	Live Agent Setup Flow
Developer Name	live_agent_setup_flow
Chat Window Title	Window Title
Allow Visitors to Save Transcripts	☐
Allow Access to Pre-Chat API	☐
Permitted Domains	
Branding Image Site	
Chat Window Branding Image	
Mobile Chat Window Branding Image	

Deployment Code

Copy this code and paste it into each web page where you want to deploy Live Agent

```
<script type='text/javascript' src='[redacted].js'></script>
<script type='text/javascript'>
liveagent.init('[redacted]', '[redacted]', '[redacted]');
</script>
```

For instance, if the deployment code contains the following information:

```
<script type='text/javascript'
       src='https://d.gla3.gus.salesforce.com/content/g/js/44.0/deployment.js'></script>
<script type='text/javascript'>
liveagent.init('https://d.gla5.gus.salesforce.com/chat', '573B00000005KXz',
'00DB00000003Rxz');
</script>
```

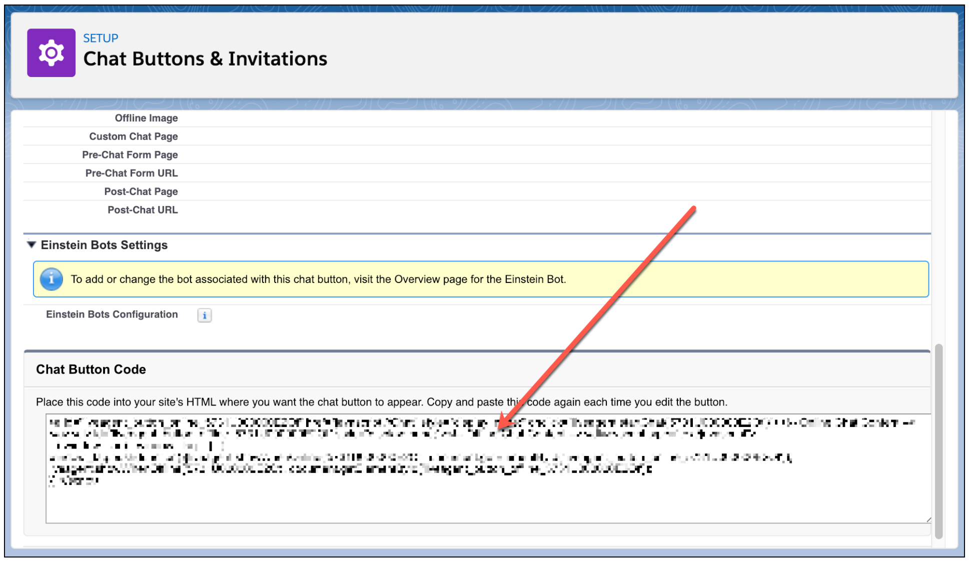
The deployment ID value is:

```
573B00000005KXz
```

Be sure not to use the org ID value (which is also in this deployment code) for the deployment ID.

Button ID

The unique button ID for your chat configuration. To get this value, from Setup, search for **Chat Buttons** and select **Chat Buttons & Invitations**. Copy the `id` for the button from the JavaScript snippet.



To add or change the bot associated with this chat button, visit the Overview page for the Einstein Bot.

Place this code into your site's HTML where you want the chat button to appear. Copy and paste this code again each time you edit the button.

For instance, if your chat button code contains the following information:

```
<a id="liveagent_button_online_575C00000004h3m"
   href="javascript://Chat"
   style="display: none;"
   onclick="liveagent.startChat('575C00000004h3m')">
   <!-- Online Chat Content -->
</a>
<div id="liveagent_button_offline_575C00000004h3m"
     style="display: none;">
     <!-- Offline Chat Content -->
</div>
<script type="text/javascript">
  if (!window._laq) { window._laq = []; }
  window._laq.push(function() { liveagent.showWhenOnline('575C00000004h3m',
    document.getElementById('liveagent_button_online_575C00000004h3m'));
    liveagent.showWhenOffline('575C00000004h3m',
    document.getElementById('liveagent_button_offline_575C00000004h3m'));
  });
</script>
```

The button ID value is:

```
575C00000004h3m
```

Be sure to omit the `liveagent_button_online_` text from the ID when using it in the SDK.

Org Setup for Chat in Salesforce Classic

To use Chat in your mobile app, first set up Chat in your org.

Note: This topic shows you how to set up Chat in Salesforce Classic. If you're using Lightning Experience, see Org Setup for Chat in Lightning Experience with a Guided Flow.

1. Create a Chat implementation in Service Cloud, as described in Chat for Administrators (PDF). Your implementation needs a deployment and a chat button.

   Note: By default, a mobile chat session times out around two minutes after you leave the app or lose connectivity. To change this value, update the **Idle Connection Timeout Duration** field when setting up your chat deployment. Keep in mind that the actual timeout on the app can be up to 40 seconds longer than the specified value in this field. See Chat Deployment Settings.

2. (Optional) If you want to use Omni-Channel for routing, configure it as described in Omni-Channel for Administrators (PDF).

   Omni-Channel enables your agents to use the same widget for all real-time routing (for example, Chat, SOS, email, case management). However, you can use Chat without setting up Omni-Channel.

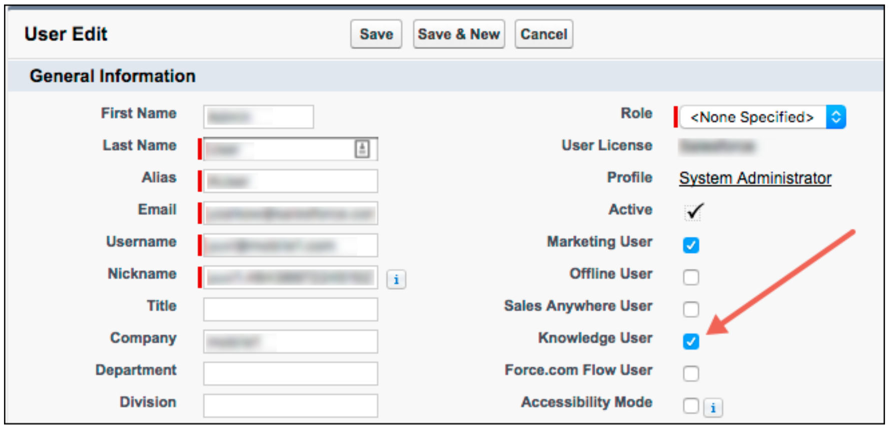
3. (Optional) If you want to build a chatbot to complement your chat experience, see Einstein Bots in Salesforce Help. In broad strokes, you must enable Einstein Bots, deploy the bot to your channel, and activate the bot. If you want to learn about building a more robust bot, see the Einstein Bots Developer Cookbook.

If you have trouble finding the settings that a developer requires to use this feature in the SDK, see Get Chat Settings from Your Org.

Cloud Setup for Knowledge

To use Knowledge in your mobile app, enable it in your org, create knowledge articles, and set up an Experience Cloud.

1. Enable Salesforce Knowledge and verify that you have Knowledge licenses. To learn more, see Enable Salesforce Knowledge in Salesforce Help.
2. In the user settings for those users you choose to administer the knowledge base, select **Knowledge User**.



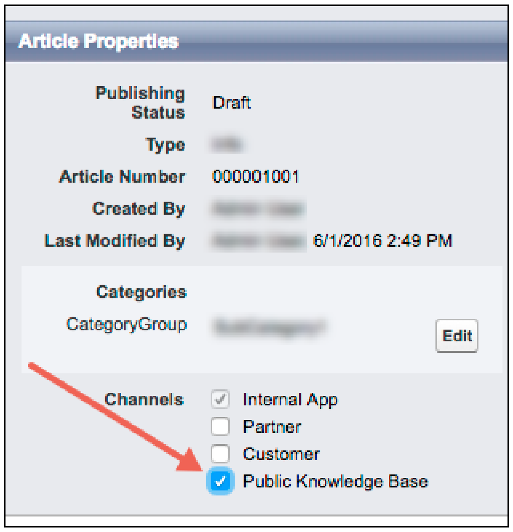
General Information

First Name	[redacted]	Role	<None Specified>
Last Name	[redacted]	User License	[redacted]
Alias	[redacted]	Profile	System Administrator
Email	[redacted]	Active	✓
Username	[redacted]	Marketing User	✓
Nickname	[redacted]	Offline User	☐
Title		Sales Anywhere User	☐
Company	[redacted]	Knowledge User	✓
Department		Force.com Flow User	☐
Division		Accessibility Mode	☐

3. Create Knowledge articles. When building out your knowledge base, make sure that you define the article types and associate articles with data categories within a category group.

   To learn more about setting up your Knowledge articles, check out: Salesforce Knowledge Documentation (HTML, PDF).

   When creating articles, ensure that they are accessible to the Public Knowledge Base channel.



Publishing Status	Draft
Type	
Article Number	000001001
Created By	
Last Modified By	6/1/2016 2:49 PM
Categories	
CategoryGroup	
Channels	✓ Internal App
	☐ Partner
	☐ Customer
	✓ Public Knowledge Base

4. Create an Experience Cloud or Salesforce site. Your Salesforce org must have an available Experience Cloud or Salesforce site. Your app developer needs the site URL for the site to use the Knowledge or Case Management feature in the SDK.

   If you've never set up a site, see Set Up and Manage Experience Cloud Sites.

5. To show Knowledge articles from your app, enable guest user access for the **Article Types**, **Categories**, and **Fields** associated with your knowledge articles. Also ensure that **Guest Access to the Support API** is turned on.

   Note: If the Guest user profile isn't set up properly, your Knowledge categories and articles do not appear.

   For step-by-step instructions, see Guest User Access for Your Experience Cloud Site.

If you have trouble finding the settings that a developer requires to use this feature in the SDK, see Get Knowledge Settings from Your Org.

Guest User Access for Your Experience Cloud Site
Ensure that guest user access is set up correctly for your Experience Cloud site. To show Knowledge articles from your app, enable guest user access for the **Article Types**, **Categories**, and **Fields** associated with your knowledge articles. To show Case Publisher, ensure that your **Quick Actions** are accessible.

Get Knowledge Settings from Your Org
After you've set up your knowledge base and your Experience Cloud site, supply your app developer with the values for the site URL, data category group, and root data category. You can get this information from your org's setup.

Guest User Access for Your Experience Cloud Site

Ensure that guest user access is set up correctly for your Experience Cloud site. To show Knowledge articles from your app, enable guest user access for the **Article Types**, **Categories**, and **Fields** associated with your knowledge articles. To show Case Publisher, ensure that your **Quick Actions** are accessible.

These instructions describe how to enable guest user access for either an Experience Cloud or Salesforce site.

1. If you're editing the settings for an Experience Cloud site:
   a. From Setup, enter `Digital Experiences` in the Quick Find box, then select **All Sites**.
   b. Click **Workspaces** for the site you'd like to access to make sure it's active.

c. From the Experience Workspaces page, select **Administration**.

d. Select **Pages** > **Go to Force.com** to get to the **Site Detail** page.

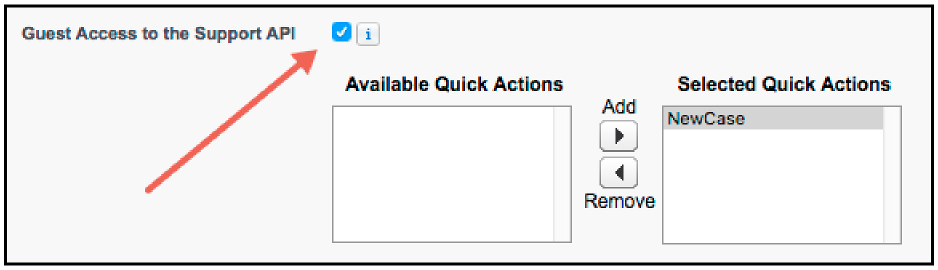
2. (Salesforce sites only) If you're editing the settings for a Salesforce site:

a. From Setup, select **Develop** > **Sites**.

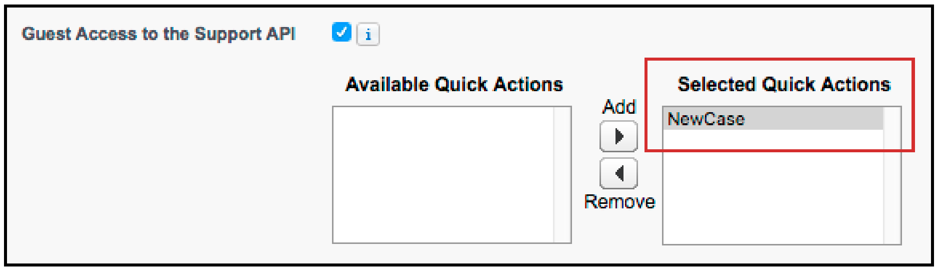
b. Click the **Site Label** for your site to get to the **Site Detail** page.

3. From the **Site Detail** section, click **Edit**.

a. Ensure that **Guest Access to the Support API** is checked.



b. (For Case Management feature only) Ensure that the desired **Quick Actions** are selected. The global quick action determines which fields display when creating a case.



c. Click **Save**.

4. (For Knowledge feature only) From the **Site Detail** section, click **Public Access Settings**. This action displays the settings for the Guest user profile in your org.

a. Verify that the user has read access to the Article Type from the **Article Type Permissions** section.

b. Verify that the user has read access to the fields in the Article Type from the **Field-Level Security** section.

c. Verify that the user has visibility to the categories from the **Category Group Visibility Settings** section.

Get Knowledge Settings from Your Org

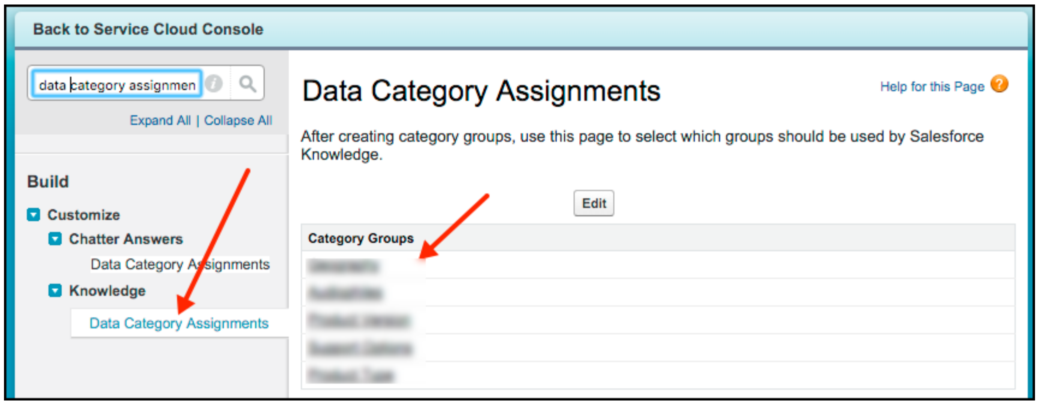
After you've set up your knowledge base and your Experience Cloud site, supply your app developer with the values for the site URL, data category group, and root data category. You can get this information from your org's setup.

Experience Cloud site URL

From Setup, search for **All Sites**, and copy the URL for the desired site.

Data Category Group

From Setup, search for **Data Category Assignments** inside the Knowledge section, and copy the name of the desired data category group.


Back to Service Cloud Console
data category assignmen
Expand All | Collapse All
Build
Customize
Chatter Answers
Data Category Assignments
Knowledge
Data Category Assignments
Help for this Page
Edit

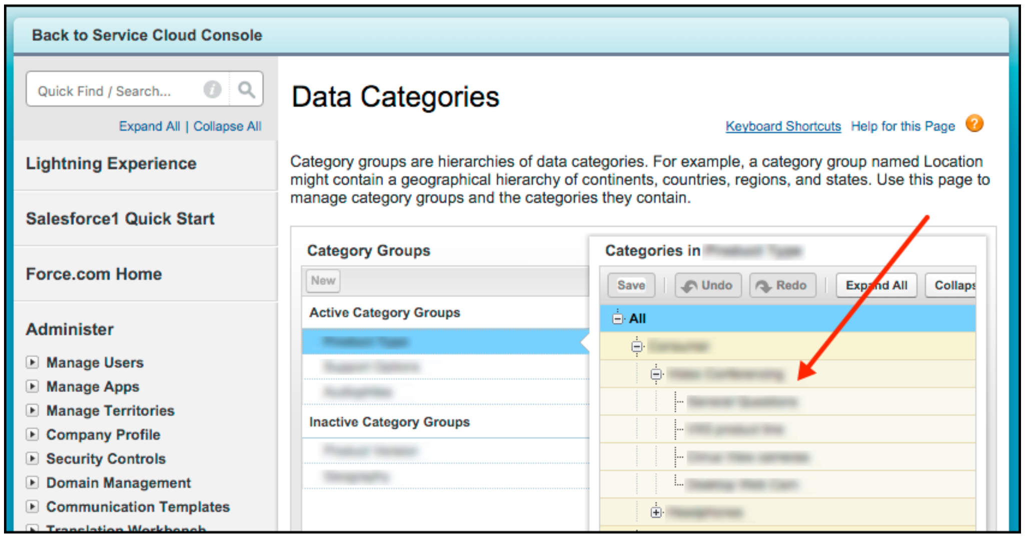
Data Category Assignments

After creating category groups, use this page to select which groups should be used by Salesforce Knowledge.

Category Groups

Data Category

From Setup, search for **Data Category Assignments** inside the Knowledge section, select the data category group, and copy the name for the desired root data category.



Data Categories

Category groups are hierarchies of data categories. For example, a category group named Location might contain a geographical hierarchy of continents, countries, regions, and states. Use this page to manage category groups and the categories they contain.

Category Groups

Active Category Groups

Inactive Category Groups

Categories in

Cloud Setup for Case Management

To use Case Management in your mobile app, set up an Experience Cloud site and create a quick action.

1. Create an Experience Cloud or Salesforce site. Your Salesforce org must have an available Experience Cloud or Salesforce site. Your app developer needs the site URL for the site to use the Knowledge or Case Management feature in the SDK.

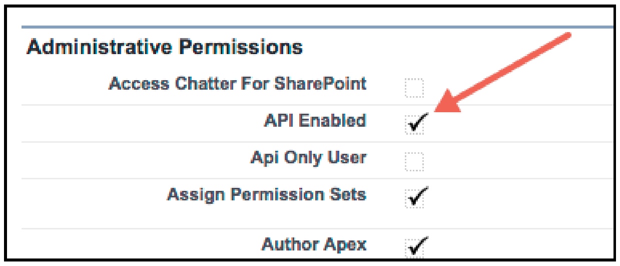
   If you've never set up a site, see Set Up and Manage Experience Cloud Sites.

2. When setting up the site, add the **Quick Actions** that you'd like to use in your app for the Case Management functionality. You must specify a quick action to use Case Management. The global quick action determines which fields display when creating a case. To learn more about quick actions, see Create Global Quick Actions in Salesforce Help. Also ensure that **Guest Access to the Support API** is turned on.

   For step-by-step instructions, see Guest User Access for Your Experience Cloud Site.

Note: Be sure that your global action is accessible to the Guest user profile. Also note that the case publisher screen does not respect field-level security for guest users. If you want to specify different security levels for different users, use different quick actions.

3. If you'd like to let authenticated users manage a list of their existing cases, you need to perform a few additional setup steps.
   a. Make sure that the **User Profile** for the authenticated users has **API Enabled** checked. For an overview on user profiles, see Profiles in Salesforce Help.



Administrative Permissions

Access Chatter For SharePoint	
API Enabled	✓
Api Only User	
Assign Permission Sets	✓
Author Apex	✓

   b. You'll need a list view for your cases in Service Cloud. To learn more about creating views, see Create a List View in Salesforce Help. Supply the **Case List Unique Name** for this view to your app developer.

      Note: If you use the built-in `My Cases` list view, keep in mind that it is filtered by the `Contact` field for community users and it is filtered by the `Created By` field for other user profiles. If you want a different behavior, create a new list view.

If you have trouble finding the settings that a developer requires to use this feature in the SDK, see Get Case Management Settings from Your Org.

Get Case Management Settings from Your Org
After you've configured your org, supply your app developer with the values for the Experience Cloud site URL, the global action, and the case list. You can get this information from your org's setup.

Get Case Management Settings from Your Org

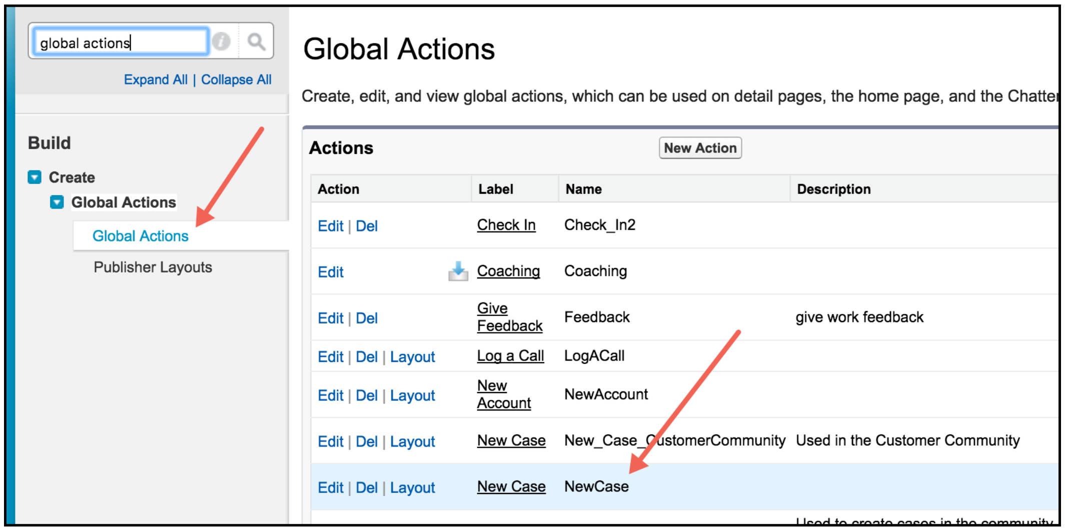
After you've configured your org, supply your app developer with the values for the Experience Cloud site URL, the global action, and the case list. You can get this information from your org's setup.

Experience Cloud site URL

From Setup, search for **All Sites**, and copy the URL for the desired site.

Global Action

From Setup, search for **Global Actions**, and copy the name of the desired quick action.



Global Actions

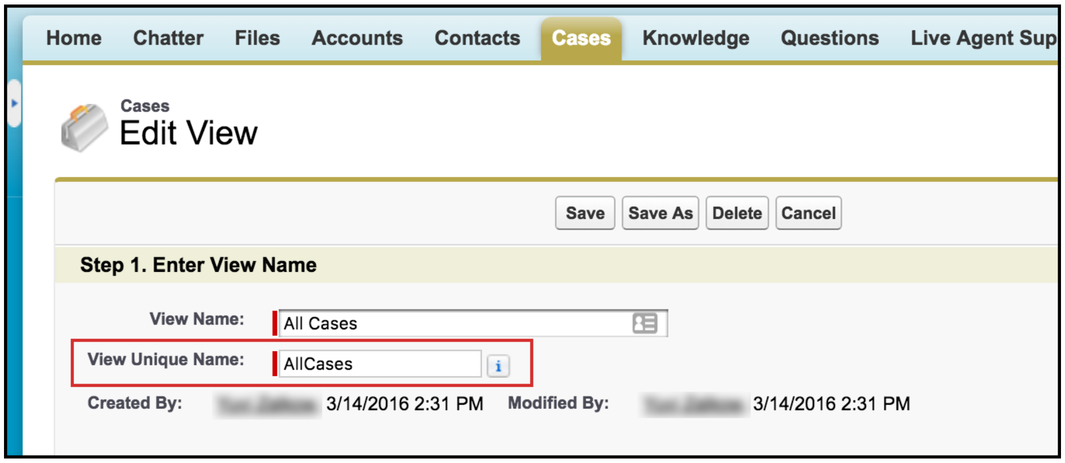
Create, edit, and view global actions, which can be used on detail pages, the home page, and the Chatte

Actions New Action

Action	Label	Name	Description
Edit | Del	Check In	Check_In2	
Edit	Coaching	Coaching	
Edit | Del	Give Feedback	Feedback	give work feedback
Edit | Del | Layout	Log a Call	LogACall	
Edit | Del | Layout	New Account	NewAccount	
Edit | Del | Layout	New Case	New_Case_CustomerCommunity	Used in the Customer Community
Edit | Del | Layout	New Case	NewCase	
			Used to create cases in the community

Case List Unique Name (for Authenticated Users Only)

To get this case list value, access the **Cases** tab in your org, pick the desired **View**, select **Go!** to see that view, and then select **Edit** to edit the view. From the edit window, you can see the **View Unique Name**. Use this value when you specify the `caseListName` in the SDK.



Cases
Edit View

Step 1. Enter View Name

View Name: All Cases
View Unique Name: AllCases
Created By: [redacted] 3/14/2016 2:31 PM Modified By: [redacted] 3/14/2016 2:31 PM

Console Setup for SOS

To use SOS in your mobile app, set up Omni-Channel and SOS for your console.

Note: If you intend to provide real-time support using *both* Chat and SOS, make sure that your agents know to go `Offline` before switching from the `Online` state of one feature to the `Online` state of the other.

To get SOS set up in your console, see Add SOS to Your Console. After setting up the SOS console, check out the other topics in this section to fine-tune your SOS configuration.

Add SOS to Your Console
Perform these steps to get SOS into your production environment.

Get SOS Settings from Your Org
After you've set up SOS in the console, supply your app developer with three values: the Chat endpoint, the organization ID, and the deployment ID. You can get this information from your org's setup.

Assign SOS Permissions
To allow an agent to use SOS, verify that the license and permissions settings are correct in Salesforce.

Automatic SOS Case Pop
With auto case pop, Service Cloud automatically creates a case when a new SOS session starts. Creating a case at the start of a session requires a trigger, a Visualforce page, and changes to the SOS session page layout.

Listen for SOS Console Events
Listen for SOS events from the Salesforce console to log activity, debug issues, and perform quality-of-service (QoS) analysis.

Record SOS Sessions
Enable SOS session recording to assure quality and let agents refer to session recordings.

SOS Reference ID
Provide an ID to give to support when there are issues with a session.

Multiple SOS Queues
Implement multiple SOS queues to route requests to specific agents or give specific requests a higher priority.

Add SOS to Your Console

Perform these steps to get SOS into your production environment.

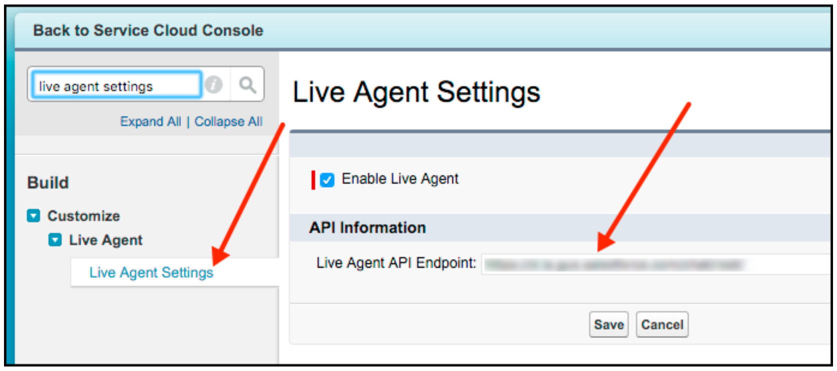
1. Configure Omni-Channel, as described in Omni-Channel for Administrators (PDF).
2. Set up SOS in Service Cloud, as described in Set Up SOS Video Chat and Screen-Sharing.
3. Be sure that you've assigned agent permissions to users, as described in Assign SOS Permissions.
4. Perform any additional customizations specified in Console Setup for SOS.

Get SOS Settings from Your Org

After you've set up SOS in the console, supply your app developer with three values: the Chat endpoint, the organization ID, and the deployment ID. You can get this information from your org's setup.

pod

The hostname for the Chat endpoint that your organization has been assigned. To get this value, from Setup, search for **Chat Settings** and copy the hostname from the **Chat API Endpoint**. Be sure not to include the protocol or the path — use only the hostname.



For instance, if your Chat API Endpoint is:

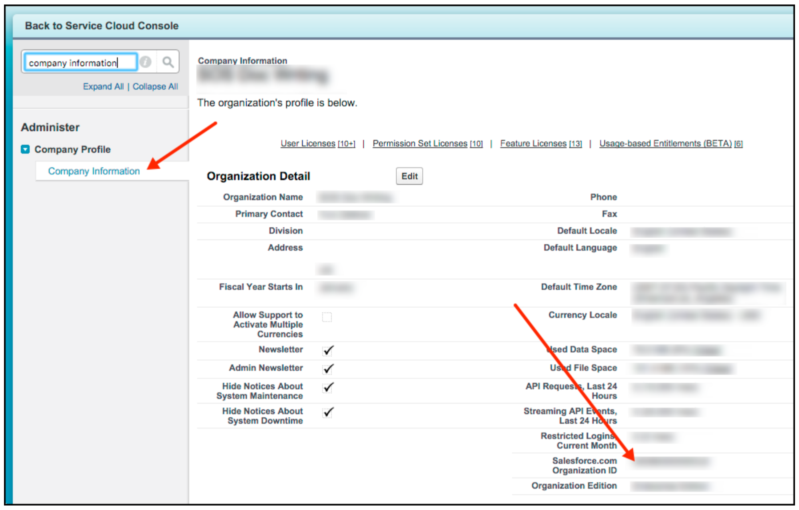
```
https://d.gla5.gus.salesforce.com/chat/rest/
```

Your pod hostname is:

```
d.gla5.gus.salesforce.com
```

orgId

The Salesforce org ID. To get this value, from Setup, search for **Company Information** and copy the **Salesforce Organization ID**.



Back to Service Cloud Console

company information

Expand All | Collapse All

Administer

Company Profile

Company Information

Company Information

The organization's profile is below.

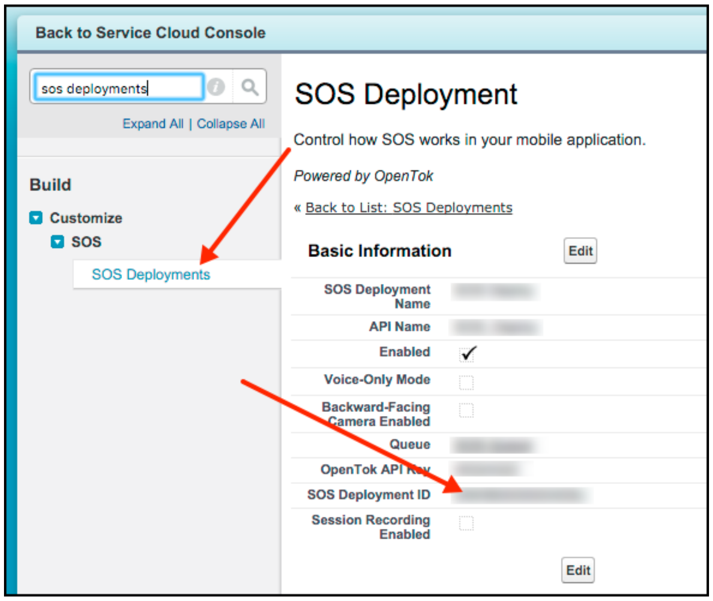
User Licenses [10+] | Permission Set Licenses [10] | Feature Licenses [13] | Usage-based Entitlements (BETA) [6]

Organization Detail Edit

Organization Name		Phone	
Primary Contact		Fax	
Division		Default Locale	
Address		Default Language	
Fiscal Year Starts In		Default Time Zone	
Allow Support to Activate Multiple Currencies	☐	Currency Locale	
Newsletter	✓	Used Data Space	
Admin Newsletter	✓	Used File Space	
Hide Notices About System Maintenance	✓	API Requests, Last 24 Hours	
Hide Notices About System Downtime	✓	Streaming API Events, Last 24 Hours	
		Restricted Logins, Current Month	
		Salesforce.com Organization ID	
		Organization Edition	

deploymentId

The unique ID of your SOS deployment. To get this value, from Setup, search for **SOS Deployments**, click the correct deployment and copy the **Deployment ID**.



Back to Service Cloud Console

sos deployments

Expand All | Collapse All

Build

Customize

SOS

SOS Deployments

SOS Deployment

Control how SOS works in your mobile application.

Powered by OpenTok

« Back to List: SOS Deployments

Basic Information Edit

SOS Deployment Name	
API Name	
Enabled	✓
Voice-Only Mode	☐
Backward-Facing Camera Enabled	☐
Queue	
OpenTok API Key	
SOS Deployment ID	
Session Recording Enabled	☐

Edit

Assign SOS Permissions

To allow an agent to use SOS, verify that the license and permissions settings are correct in Salesforce.

1. Assign an SOS user license.

   Assigning a license must be done for every user that requires access to SOS.

   a. From **Setup**, select **Manage Users** > **Users**.
   b. Click the name of the user. (Do not click **Edit**.)
   c. Select **Permission Set License Assignments** and then click **Edit Assignments**.
   d. Enable **SOS User**. If this option is not available, your org has not been assigned any SOS licenses.
   e. Click **Save**.

2. Enable the SOS license.

   Once licenses are assigned to users, enable them using a permission set. We recommend that you have a permission set specifically for SOS, because all users assigned to this permission set must have an SOS license. If you attempt to enable SOS for a permission set which contains users that do not have an SOS license, you'll receive an error.

   a. From **Setup**, select **Manage Users** > **Permission Sets**.
   b. If you do not have a permission set for SOS, click the **New** button. Give it a **Label** and click **Save**.
   c. If you already have a permission set, click the SOS permission set.
   d. Click **App Permissions** and then click the **Edit** button.
   e. Check **Enable** for the **Enable SOS Licenses** checkbox. You'll receive an error if any assigned users do not have the SOS license.
   f. Click **Save**.

3. Enable the service presence status.

   You can enable the service presence status using either permission sets or profiles. If the presence status is only being used for SOS, it is easier to enable the presence status through the same permission set that enables the license. Using the same permission set guarantees that all agents who require the presence status have access to it. If the presence status is being used for multiple service channels, it is likely that the same permission set cannot be used, since all members of the permission set would require a SOS license. In this case, you may want to have multiple permissions sets, assign it to a profile, and use some combination of profiles and permission sets.

   Service Permission via Permission Sets

   a. From **Setup**, select **Manage Users** > **Permission Sets**.
   b. Click an existing permission set associated with SOS, or create a new one.
   c. Click **Service Presence Statuses Access** and then click the **Edit** button.
   d. Add the service presence related to SOS to the **Enabled Service Presence Statuses**.
   e. Click **Save**.
   f. If necessary, click **Manage Assignments** to add agents to the permission set.

   Service Permission via Profile

   a. From **Setup**, select **Manage Users** > **Profiles**.
   b. Click the name of the profile associated with SOS. (Do not click **Edit**.)
   c. Click the **Edit** button for **Enabled Service Presence Status Access**.
   d. Add the service presence related to SOS to the **Enabled Service Presence Statuses**.
   e. Click **Save**.

4. Add agents to the queue.

All agents must be a member of at least 1 queue. You can determine which queues are used by SOS by looking at the SOS deployments. Agents can be added to a queue individually or in groups. These groups differ depending on the org — groups can be broken into: roles, public groups, partner users, and so on.

a. From **Setup**, select **Manage Users** > **Queues**.

b. Click **Edit** for the desired queue.

c. Scroll to the bottom of the page and find the **Queue Members** section. Add the required members.

d. Click **Save**.

Automatic SOS Case Pop

With auto case pop, Service Cloud automatically creates a case when a new SOS session starts. Creating a case at the start of a session requires a trigger, a Visualforce page, and changes to the SOS session page layout.

1. Create a trigger.

   This trigger fires before a new SOS session object saves. The trigger creates a case and adds a reference to the case to the SOS session object. When the case is created, the owner is initially set to "Automated Process". This value changes to the owner of the SOS session object with the Visualforce page specified in the next step.

   a. From **Setup**, search for **SOS Sessions**.

   b. Select **Triggers** from the **SOS Sessions** section.

   c. Click the **New** button.

   d. Replace the **Apex Trigger** code with the code below. This code assumes that the email address is sent through the **SOS Custom Data** feature using the `Email__c` API Name. To learn more about custom data in SOS, see Using SOS with the Service SDK. Any data that can be used to identify a contact can be sent instead of the email, as long as the trigger is updated to reflect this information.

```
trigger SOSCreateCaseCustom on SOSSession (before insert) {
  List<SOSSession> sosSess = Trigger.new;
  for (SOSSession s : sosSess) {
    try {
      Case caseToAdd = new Case();
      caseToAdd.Subject = 'SOS Video Chat';
      if (s.ContactId != null) {
        caseToAdd.ContactId = s.ContactId;
      } else {
        List<Contact> contactInfo =
            [SELECT Id from Contact WHERE Email = :s.Email__c];
        if (!contactInfo.isEmpty()) {
          caseToAdd.ContactId = contactInfo[0].Id;
          s.ContactId = contactInfo[0].Id;
        }
      }
      insert caseToAdd; s.CaseId = caseToAdd.Id;
    }
    catch(Exception e){}
  }
}
```

   e. Click **Save**.

2. Add a Visualforce page.

   This Visualforce page changes the owner of the case and opens the case in a subtab. The page is added to the SOS session page layout in the final step.

   a. From **Setup**, search for **Visualforce Pages**.
   b. Click the **New** button.
   c. Give the Visualforce page a name. For example, "SOS_Open_Case_Custom".
   d. Replace the **Visualforce Markup** code with this code:

```
<apex:page sidebar="false" standardStylesheets="false">
  <apex:includeScript value="/soap/ajax/34.0/connection.js"/>
  <apex:includeScript value="/support/console/34.0/integration.js"/>

  <script type='text/javascript'>
    sforce.connection.sessionId = '{!$Api.Session_ID}';

    function escapeSoql (str) {
      return str.replace(/\\/g, '\\\\').replace(/'/g, "\\'");
    }

    document.addEventListener('DOMContentLoaded', function () {
      sforce.console.getEnclosingPrimaryTabObjectId(function(result) {
        if (!result || !result.success) {
          return;
        }

        var sosSessionId = result.id;
        var query =
            "SELECT CaseId, OwnerId FROM SOSSession WHERE Id = '" +
             escapeSoql(sosSessionId) + "'"
        var queryResult = sforce.connection.query(query);
        var record = queryResult.getArray('records');

        if (!record || !record[0]) {
          console.log('Can not determine session Id');
          return;
        }

        var caseId = record[0].CaseId;
        var ownerId = record[0].OwnerId;

        if (!ownerId) {
          console.log('No owner Id');
          return;
        }

        var caseUpdate = new sforce.SObject("Case");
        caseUpdate.Id = caseId;
        caseUpdate.OwnerId = ownerId;
        result = sforce.connection.update([caseUpdate]);

        if (!result[0].getBoolean("success")) {
          console.log('Unable to set owner', result, caseUpdate);
        }

        sforce.console.getEnclosingPrimaryTabId(function(result) {
          if (!result || !result.success) {
            return;
          }

          var query = "SELECT CaseNumber FROM Case WHERE Id = '" +
                       escapeSoql(caseId) + "'"
          var queryResult = sforce.connection.query(query);
          var record = queryResult.getArray('records');
          var caseNumber = record && record[0] &&
                           record[0].CaseNumber || 'Case';

          sforce.console.openSubtab(result.id, '/'+caseId,
                                    true, caseNumber);
        });
      });
    });
  </script>
</apex:page>
```

e. Click **Save**.

3. Update the SOS session page layout.

   Now that the Visualforce page has been created, you can change the page layout of the SOS session. This change hides the Visualforce page in the layout.

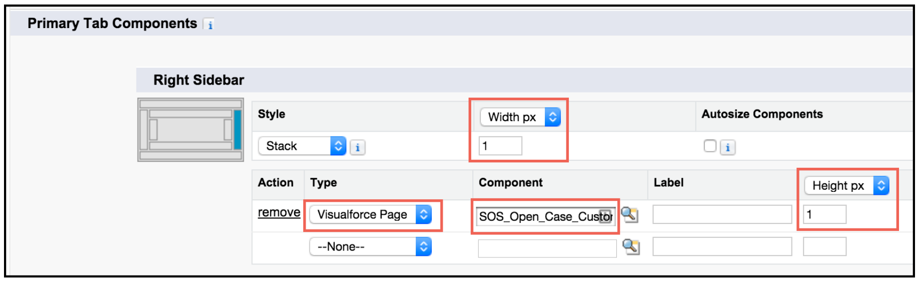
   a. From **Setup**, search for **SOS Sessions**.

   b. Select **Page Layouts** from the **SOS Sessions** section.

   c. Click **Edit** for your active layout (probably **SOS Session Layout**).

   d. From the top of the page, select the **Custom Console Components** link.

   e. Under the **Primary Tab Components** section, add the following to one of the sidebars:



Primary Tab Components

Right Sidebar

Style	Width px	Autosize Components
Stack	1	☐

Action	Type	Component	Label	Height px
remove	Visualforce Page	SOS_Open_Case_Custo		1
	--None--			

   - Set **Style** to **Stack**.
   - Set **Width px** to **1**.
   - Set **Height px** to **1**. (Change **Height %** to **Height px** if necessary.)
   - Set **Type** to **Visualforce Page**
   - Set **Component** to the page created previously (for example, **SOS_Open_Case_Custom**).

   f. Click **Save**.

Whenever an agent accepts an incoming call, a case automatically gets created.

Listen for SOS Console Events

Listen for SOS events from the Salesforce console to log activity, debug issues, and perform quality-of-service (QoS) analysis.

To detect events from the Salesforce console, use JavaScript in your Visualforce page. Call the `addEventListener` method, which is documented in the Salesforce Console Developer Guide. The method syntax is:

```
sforce.console.addEventListener(eventType: String, eventListener: Function);
```

Parameter	Type	Description
eventType	String	The event type. For SOS session state events, this value is `SFORCE_SOS:STATE_CHANGED`. For audio QoS, this value is `SFORCE_SOS:QOS_AUDIO`. For video QoS, this value is `SFORCE_SOS:QOS_VIDEO`.
eventListener	Function	This function is called when the registered event is emitted. You receive one JSON `message` object within the payload passed to this function.

For more information about `SFORCE_SOS:STATE_CHANGED`, see SOS State Change Console Events.

For more information about `SFORCE_SOS:QOS_AUDIO` and `SFORCE_SOS:QOS_VIDEO`, see SOS Quality-of-Service Console Events.

SOS State Change Console Events
You can listen for SOS session state changes from the Salesforce console for logging and debugging purposes.

SOS Quality-of-Service Console Events
You can listen for SOS audio and video quality-of-service (QoS) events from the Salesforce console.

SOS State Change Console Events

You can listen for SOS session state changes from the Salesforce console for logging and debugging purposes.

After you add an event listener for state changes to your console (see Listen for SOS Console Events), inspect your function's payload and handle the event.

```
sforce.console.addEventListener("SFORCE_SOS:STATE_CHANGED", function(payload) {
  // Handle event
});
```

Event Listener Payload

The JSON payload you receive within the event listener function follows this syntax.

```
{
  message: {
    sfdcSosSessionId: <SESSION_ID>,
    currentState: <CURRENT_STATE>,
    previousState: <PREVIOUS_STATE>,
    reason: <REASON_IF_APPLICABLE>
  }
}
```

`sfdcSosSessionId` (String)
: The ID associated with the session that emitted the events.

`currentState` (String)
: The current state of the session. See the Event States section.

`previousState` (String)
: The previous state of the session. See the Event States section.

`reason` (Object or null)
: Populated only when the current state is `ENDED`. Contains information about why the session was ended. See the End Reasons section.

This code sample illustrates how to handle an event. Subsequent sections describe how to interpret each part of the message object payload.

```
sforce.console.addEventListener("SFORCE_SOS:STATE_CHANGED",  function(event) {
  var stateChange = {};
  try {
    stateChange = JSON.parse(event.message);
  } catch (e) {
    // Error Parsing JSON Object
    throw new Error(e);
  }

  /*
    Use currentState vs previousState to determine how you reached
    the state you're in. Most useful in the case of the ENDED state
    where you want to know how it ended and if there were any errors.
  */

  var currentState = stateChange.currentState;
  var previousState = stateChange.previousState;

  // Handle Non ENDED state changes
  if (currentState !== 'ENDED') {
    logStateChange(currentState, previousState);
    return;
  }

  // Handle ENDED state change
  switch (stateChange.reason.name) {

    // Handle a session that was intentionally ended by customer or agent
    case 'ENDED_BY_CUSTOMER':
    case 'ENDED_BY_CONSOLE':
      logEndedSession(currentState, previousState, stateChange.reason.name);
      break;

    // Handle a session that ended in an error
    case 'ERROR':
      logEndedWithError(currentState, previousState, stateChange.reason.name,
stateChange.reason.error);
      break;
  }
});
```

Event States

The `currentState` and `previousState` fields can be one of the following states.

`LOADING_RESOURCES`
Widget has loaded. Fetching more resources from the server.

`INTERFACE_CHECK`
Applies to Internet Explorer browsers only. Attempting to install the Internet Explorer plug-in.

`JOINING`
Joining the audio/video session.

`INITIALIZING`
Starting to listen for updates from the audio/video session.

`AV_CONNECTION`
Connecting to the audio/video session, getting microphone or camera permissions from the browser, and starting to send the stream to the audio/video session.

`WAITING`
Waiting to receive the audio/video stream from the SDK.

`CONNECTED`
Session is fully established with both audio and video.

`HOLD`
Session has been put on hold by the customer, the agent, or both.

`PAUSED`
Session paused because the app was put into the background, the customer is typing into a masked field, or the customer accepted a phone call.

`ENDED`
The session has completed. See the End Reasons section for more information.

End Reasons

When the `currentState` is `ENDED`, you can inspect the `reason` object to find out why the session ended.

```
{
  message: {
    sfdcSosSessionId: <SESSION_ID>,
    currentState: <CURRENT_STATE>,
    previousState: <PREVIOUS_STATE>,

    reason: {
      name: <END_REASON>
      error: <ERROR_IF_APPLICABLE>
    }
  }
}
```

`name` (String)
The reason why the session ended. Can be `ENDED_BY_CUSTOMER`, `ENDED_BY_CONSOLE`, or `ERROR`.

`error` (Error object or null)
If there's an error, this field contains the error details. If there isn't an error, the value is `null`. See Errors section.

The following table describes various ways a session can end, with and without an error. See the Errors section for details about session failures.

Table 1: End Reason Scenarios

State	How a Session Can End Without an Error	How a Session Can End with an Error
LOADING_RESOURCES	The customer or agent manually ended the session prematurely.	An issue occurred while loading scripts from the server.
INTERFACE_CHECK	The customer or agent manually ended the session prematurely. Possibly related to a user issue while installing the plug-in.	The agent encountered an issue while installing the OpenTok plug-in.
JOINING	The customer or agent manually ended the session prematurely.	The agent encountered an issue while joining the SOS session.
INITIALIZING	The customer or agent manually ended the session prematurely.	The agent encountered an issue while starting to listen for updates from the SOS session.
AV_CONNECTION	The customer or agent manually ended the session prematurely.	The agent encountered an issue while joining the session. The issue could be due to hardware permissions, misconfigured firewall rules, network performance, or an internal server error.
WAITING	The customer disconnected from the session without ending it, causing the session to end. Typical reasons include the customer lost network connectivity, the app crashed, or the customer closed the app.	The agent failed to connect to the customer's audio/video stream. The issue could be due to misconfigured firewall rules, network performance, an internal server error, or the customer was dropped from the audio/video session.
CONNECTED	The customer or agent manually ended the session from a normal state.	The session ended unexpectedly with a fatal error. This error could be due to network performance issues or an internal server error.

State	How a Session Can End Without an Error	How a Session Can End with an Error
HOLD	The customer or agent manually ended the session from a normal state. The agent or customer could have ended the session after being in the hold state for too long.	The session ended unexpectedly with a fatal error. This error could be due to network performance or an internal server error.
PAUSED	The customer or agent manually ended the session from a normal state. The agent or customer could have ended the session after being in the paused state for too long.	The session ended unexpectedly with a fatal error. This error could be due to network performance or an internal server error.
ENDED	The session ended without issue.	The session experienced issues while disconnecting from the audio/video session or making a request to the audio/video server.

The following table shows some scenarios in which a session can end successfully, along with a sample payload.

Table 2: Successful End Reason Examples

Scenario	Sample Payload

Session ended by customer	`{`
	`  currentState: 'ENDED',`
	`  previousState: 'CONNECTED',`
	`  reason: {`
	`    name: 'ENDED_BY_CUSTOMER',`
	`    error: null`
	`  }`
	`}`

Session ended by agent	`{`
	`  currentState: 'ENDED',`
	`  previousState: 'CONNECTED',`
	`  reason: {`
	`    name: 'ENDED_BY_CONSOLE',`
	`    error: null`
	`  }`
	`}`

Session ended by agent while session is on hold	`{`
	`  currentState: 'ENDED',`
	`  previousState: 'HOLD',`
	`  reason: {`
	`    name: 'ENDED_BY_CONSOLE',`
	`    error: null`
	`  }`
	`}`

Scenario	Sample Payload

Session ended by agent while customer app is in the background	
	```
	{
	  currentState: 'ENDED',
	  previousState: 'PAUSED',
	  reason: {
	    name: 'ENDED_BY_CONSOLE',
	    error: null
	  }
	}
	```

Session ended by agent after app crash	
	```
	{
	  currentState: 'ENDED',
	  previousState: 'WAITING',
	  reason: {
	    name: 'ENDED_BY_CONSOLE',
	    error: null
	  }
	}
	```

Errors

When a session ends with an error, inspect the `error` object for more information. The error syntax is:

```
{
  message: {
    sfdcSosSessionId: <SESSION_ID>,
    currentState: <CURRENT_STATE>,
    previous: <PREVIOUS_STATE>,
    reason: {
      name: 'ERROR',

      error: {
        code: <ERROR_CODE>,
        domain: <ERROR_DOMAIN>,
        message: <ERROR_MESSAGE>,
        name: <ERROR_NAME>,
        type: <ERROR_TYPE>,
        rawError: {
          code: <RAW_ERROR_CODE>,
          message: <RAW_ERROR_MESSAGE>,
          name: <RAW_ERROR_MESSAGE>
        }
      }
    }
  }
}
```

`code` (Number)

Error code used for grouping related errors. Some common error codes include: 1000 (SOS session timed out waiting to access camera or microphone); 1001 (audio/video request timed out), 1003 (failed to set agent name); 1006 (SOS session timed out waiting to access the camera or microphone); 1500 (permission to audio/video hardware denied). See OpenTok's Handling Exceptions documentation for more error conditions.

`domain` (String or null)
Describes the category of error when it's related to an OpenTok audio/video issue. Can be one of the following: `session`, `publisher`, or `subscriber` domains. A `session` error relates to an existing audio/video session. A `publisher` error describes an issue that the agent had when creating an audio/video stream. A `subscriber` error describes an issue that the agent had when receiving a customer's audio/video stream.

`message` (String)
A description of what caused the error.

`name` (String or null)
A unique name associated with the error.

`type` (String)
Specifies from where the error originated. Can be one of the following: `opentok` (the underlying WebRTC platform); `scrt` (Salesforce's real-time server); or `widget` (the Salesforce console widget).

`rawError` (Object)
The raw error returned by the server without parsing, grouping, or renaming.

The following table shows some scenarios in which a session can end in an error, along with a sample payload.

Table 3: End in Error Examples

Error Scenario	Sample Payload

Agent declines permissions prompt	
	```
	{
	  currentState: 'ENDED',
	  previousState: 'AV_CONNECTION',
	  reason: {
	    name: 'ERROR',
	    error: {
	      code: 1500,
	      domain: 'publisher',
	      message: 'Permission to audio/video
	hardware
	        denied. You must grant permission
	for SOS to
	        access microphone and camera.',
	      name: 'OT_USER_MEDIA_ACCESS_DENIED',
	      type: 'opentok',
	      rawError: {
	        code: 1500,
	        message: 'ORIGINAL ERROR MESSAGE',
	       name: 'OT_USER_MEDIA_ACCESS_DENIED'

	      }
	    }
	  }
	}
	```

Agent lets session time out without granting permissions	
	```
	{
	  currentState: 'ENDED',
	```

Error Scenario	Sample Payload

	`  previousState: 'AV_CONNECTION',`
	`  reason: {`
	`    name: 'ERROR',`
	`    error: {`
	`      code: 1000,`
	`      domain: null,`
	`      message: 'SOS session timed out`
	`waiting to`
	`        access camera/microphone.',`
	`      name: null,`
	`      type: 'widget',`
	`      rawError: {`
	`        code: 1000,`
	`        message: 'SOS session timed out`
	`waiting to`
	`          access camera/microphone.'`
	`      }`
	`    }`
	`  }`
	`}`

Session dies after OpenTok drops connection because of a timeout	`{`
	`  currentState: 'ENDED',`
	`  previousState: 'AV_CONNECTION',`
	`  reason: {`
	`    name: 'ERROR',`
	`    error: {`
	`      code: 1006,`
	`      domain:session,`
	`      message: 'SOS session timed out`
	`waiting to`
	`        access camera/microphone.',`
	`      name: 'OT_SOCKET_CLOSE_ABNORMAL',`
	`      type: 'opentok',`
	`      rawError: {`
	`        code: 1006,`
	`        message: 'Unable to connect to the`
	`session.`
	`          Please ensure you have network`
	
	`          connectivity.',`
	`        name: 'OT_SOCKET_CLOSE_ABNORMAL'`
	`      }`
	`    }`
	`  }`
	`}`

SOS Quality-of-Service Console Events

You can listen for SOS audio and video quality-of-service (QoS) events from the Salesforce console.

Note: The console allows you to track streaming issues on the other side of the conversation (from the client to the OpenTok media router). To track QoS issues on this side (from the agent to the media router), refer to the SOS SDK documentation on quality-of-service events: Using SOS with the Service SDK.

After you add an event listener for QoS to your console (see Listen for SOS Console Events), inspect your function's payload and handle the event.

```
sforce.console.addEventListener("SFORCE_SOS:QOS_AUDIO", function(payload) {
  // Handle audio QoS event
});
sforce.console.addEventListener("SFORCE_SOS:QOS_VIDEO", function(payload) {
  // Handle video QoS event
});
```

Audio QoS Event Listener Payload

This sample JSON payload is for the `SFORCE_SOS:QOS_AUDIO` event type.

```
{
  "message":"{
    "bytesReceived":131131,
    "packetsLost":3,
    "packetsReceived":1499,
    "timestamp":1502214189391,
    "sfdcSosSessionId":"0NXR000000000MS"
  }"
}
```

This payload specifies how many bytes were received, the number of packets lost, and the number of packets received for a 30-second span. If the session ends before 30 seconds, QoS data isn't logged.

Video QoS Event Listener Payload

This sample JSON payload is for the `SFORCE_SOS:QOS_VIDEO` event type.

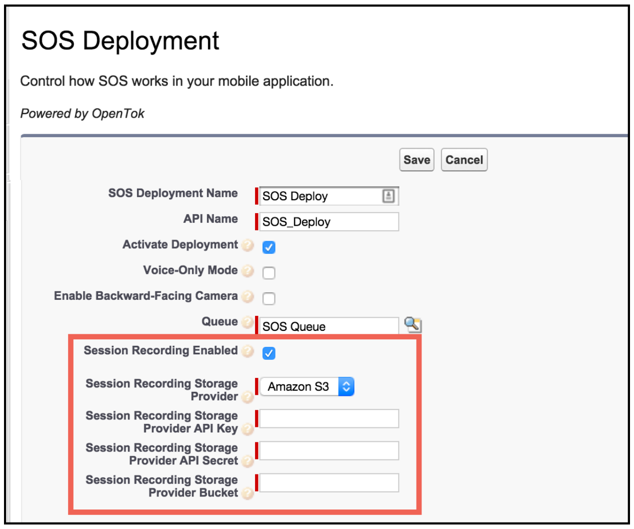
```
{
  "message":"{
    "bytesReceived":82253,
    "packetsLost":0,
    "packetsReceived":337,
    "timestamp":1502214189391,
    "size":"480x640",
    "timePerShareType":{
      "ss":"55.29",
      "ffc":"44.71",
      "bfc":"0.00"
    },
    "sfdcSosSessionId":"0NXR000000000MS"
  }"
}
```

This payload specifies how many bytes were received, the number of packets lost, and the number of packets received for a 30-second span. If the session ends before 30 seconds, QoS data isn't logged. This payload also describes the resolution size of the video and the percentage of time in each share type. The share types are `ss` for screen sharing, `ffc` for the front-facing camera, and `bfc` for the back-facing camera.

Record SOS Sessions

Enable SOS session recording to assure quality and let agents refer to session recordings.

1. From **Setup**, search for **SOS Deployments**.
2. Select your deployment.
3. Check the **Session Recording Enabled** checkbox. Specify your API key, secret, and bucket.



SOS Deployment

Control how SOS works in your mobile application.

Powered by OpenTok

SOS Deployment Name	SOS Deploy
API Name	SOS_Deploy
Activate Deployment	✓
Voice-Only Mode	
Enable Backward-Facing Camera	
Queue	SOS Queue
Session Recording Enabled	✓
Session Recording Storage Provider	Amazon S3
Session Recording Storage Provider API Key	
Session Recording Storage Provider API Secret	
Session Recording Storage Provider Bucket	

You can retrieve recorded sessions in the mp4 format from your Amazon S3 bucket.

Note: To configure your AWS environment, see Managing Access Permissions to Your Amazon S3 Resources.

SOS Reference ID

Provide an ID to give to support when there are issues with a session.

The SOS Reference ID (also referred to as the SOS Session ID) is a unique ID used to identify a session. It is 15 characters and starts with "0NX". If there is an issue with a session, this ID can be provided to Support to locate logs related to the session.

There are two ways to find the SOS Reference ID:

1. Add it to the SOS Session object
2. Add it to the fields displayed in the SOS Session list view.

Add to Session Object

If the SOS Reference ID is added to the SOS Session Object and SOS Session page layouts, the ID can be seen when viewing any SOS Session. To add the SOS Reference ID to the SOS Session object:

1. From **Setup**, search for **SOS Sessions**.
2. From **SOS Sessions**, select **Fields**. (Do not go to Fields under SOS Session Activities.)
3. Click **New** under **SOS Session Custom Fields & Relationships**.
4. Select **Formula**. Click **Next**.
5. Enter **SOS Reference Id** as the **Field Label**. **Field Name** auto populates.

6. Select **Text** as the **Formula Return Type**. Click **Next**.
7. In the **Simple Formula** text area, enter **Id**. Click **Next**.
8. Click **Next** again. (Permission to view the field can be removed on this page before clicking next.)
9. Click **Save**.

   We recommend that you add this field to all page layouts.

Add to Session List View

The SOS Session list view can be added as a navigation tab item to any console app. Using the SOS Session list view allows you to view the SOS Reference ID for multiple sessions on a single screen. To add the SOS Session list view to a console app:

1. From **Setup**, search for **Apps**.
2. Select **Edit** for the desired console app.
3. Under **Choose Navigation Tab Items**, move **SOS Sessions** to **Selected Items**.
4. Click **Save**.

The SOS Session list view may not display the SOS Reference ID by default. If so, a view can be edited or a new view can be created. To add the SOS Reference Id to the view:

1. Go to the SOS Session list view
2. Click either **Edit** or **Create New View**.
3. Now you can determine which fields are visible.
   - If the new field was added (as shown earlier) move **SOS Reference Id** to **Selected Fields**.
   - If the new field was not created, move both instances of **SOS Session Id** to **Selected Fields**. (The two SOS Session Id fields are different fields. One is the unique ID that starts with the characters "0NX"; the other is a number that increments for each session.)
4. Click **Save**.

Multiple SOS Queues

Implement multiple SOS queues to route requests to specific agents or give specific requests a higher priority.

Multiple queues can help out in the following situations:

- Giving paying customers a higher priority
- Having separate queues for different products
- Routing to agents with specific skill sets
- Giving agents a personal queue (great for training)
- Creating a training queue that has a lower priority or only gets requests from simple pages
- Grouping separate pages into different queues

You need two objects to make multiple queues work: a `Queue` and an `SOS Deployment` object. A third object, `Routing Configuration`, lets you use different priorities.

1. If a `Routing Configuration` is being used to achieve different priorities, create this object first. If you want all queues to have the same priority, the same routing configuration can be used.
2. Next, create the `Queue`. The queue references the routing config. An agent can be a member of multiple queues.

3. Create the `SOS Deployment` last. The deployment references the queue. An app may have access to several SOS deployment IDs, and then the app decides which queue the user should be sent to using the SOS deployment ID.

Embedded Service SDK for Mobile Apps Setup

Set up the SDK to start using Service Cloud features in your mobile app.

Requirements for the Service SDK for Android

Before you set up the SDK, let's take care of a few pre-reqs.

Accessibility with the Service SDK for Android

The Service SDK is accessible to customers that use a screen reader. Depending on your needs, you can also change some settings to expand accessibility.

Install the Service SDK for Android

Install the Service SDK for Android using Gradle.

Authentication with the Service SDK for Android

The Service SDK provides an authentication mechanism that allows your users to access user-specific information in Service Cloud. To authenticate, implement two interfaces and provide an access token to the SDK.

Push Notifications with the Service SDK for Android

To take advantage of push notifications from your org to your app, set up an Apex trigger and configure your app for notifications. Pass relevant notification information, such as case feed activity, to the Service SDK using your `PushNotificationListener` implementation.

Analytics with the Service SDK for Android

You can listen to user-driven events from the Service SDK using the `ServiceAnalytics` system.

Decrease the Size of Your App

Although the SDK doesn't have a large footprint, you can decrease the size of your app by splitting your APK or by using ProGuard.

Requirements for the Service SDK for Android

Before you set up the SDK, let's take care of a few pre-reqs.

Salesforce Org Requirements

The Service SDK can be used with both Lightning Experience and Salesforce Classic. However, the SOS agent widget currently only works in Salesforce Classic.

SDK Requirements

This SDK requires Android API level 21 (5.0, Lollipop) or newer. You can access the SDK using either Java or Kotlin.

App Permission Requirements

Depending on which features you use, the SDK inserts permission requirements into the manifest of your compiled application package. The following table describes which requirements are added for which features.

Permission	Knowledge	Case Management	Chat	SOS
General Purpose Permissions				
android.permission.ACCESS_NETWORK_STATE	Yes	Yes	Yes	Yes
android.permission.INTERNET	Yes	Yes	Yes	Yes
android.permission.READ_EXTERNAL_STORAGE	Yes	Yes	Yes	Yes
android.permission.WRITE_EXTERNAL_STORAGE	Yes	Yes	Yes	Yes
Authentication-Related Permissions				
android.permission.AUTHENTICATE_ACCOUNTS	Yes (auth only)	Yes (auth only)		
android.permission.GET_ACCOUNTS	Yes (auth only)	Yes (auth only)		
android.permission.MANAGE_ACCOUNTS	Yes (auth only)	Yes (auth only)		
android.permission.USE_CREDENTIALS	Yes (auth only)	Yes (auth only)		
android.permission.USE_FINGERPRINT	Yes (auth & API 23+)	Yes (auth & API 23+)		
android.permission.WAKE_LOCK	Yes (auth only)	Yes (auth only)		
com.google.android.c2dm.permission.RECEIVE	Yes (auth only)	Yes (auth only)		
Real-Time Chat and Video Permissions				
android.permission.BLUETOOTH				Yes
android.permission.BROADCAST_STICKY				Yes
android.permission.CAMERA			Yes	Yes
android.permission.MODIFY_AUDIO_SETTINGS				Yes
android.permission.READ_PHONE_STATE				Yes (API 22 & earlier)
android.permission.RECORD_AUDIO				Yes

The permissions that specify "auth only" are required if you're using authenticated user accounts. If you're not using authenticated accounts, you can manually remove these permissions from the manifest with the `tools:node` attribute. For example:

```
<uses-permission android:name="android.permission.AUTHENTICATE_ACCOUNTS"
tools:node="remove"/>
```

SOS Agent Requirements

The agents responding to SOS calls must have modern browsers and reasonably high-speed internet connectivity to handle the demands of real-time audio and video.

Hardware requirements:

- Webcam
- Microphone

Bandwidth requirements:

- 500 Kbps upstream
- 500 Kbps downstream

Important: Due to bandwidth limitations, 2G networks, such as GPRS and EDGE, are not supported.

Browser requirements:

- Chrome version 35 or newer
- Firefox version 30 or newer
- Internet Explorer version 10 or newer (plug-in required)

Note: Your browser must support Transport Layer Security (TLS) protocol version 1.1 or newer. If you are running Internet Explorer version 10, see this help topic on how to update the TLS version.

Operating System:

- OSX 10.5 or newer
- Windows 7 or newer

Accessibility with the Service SDK for Android

The Service SDK is accessible to customers that use a screen reader. Depending on your needs, you can also change some settings to expand accessibility.

Disable the Minimized View in Chat

By default, a chat session starts out as a minimized, thumbnail view that you tap to open. This minimized view is not optimal for accessibility because a visually impaired person could have trouble locating the thumbnail. To improve accessibility, we suggest starting the session in the full-screen view. When building the `ChatUIConfiguration` object, set `defaultToMinimized` to `false`.

```
ChatUIConfiguration uiConfig = new ChatUIConfiguration.Builder()
  .chatConfiguration(chatConfiguration)
  .defaultToMinimized(false)
  .build();
```

See Quick Setup: Chat in the Service SDK.

Contrast Ratio Considerations

By default, we brand the SDK using a 4.2 contrast ratio. You can customize the colors to increase this contrast ratio.

See Customize Colors with the Service SDK.

Known Issues

- Chat: A screen reader doesn't read the event text when an agent invites another agent to the chat session.

Install the Service SDK for Android

Install the Service SDK for Android using Gradle.

Before running through these steps, be sure you've checked the Requirements for the Service SDK for Android.

To get started with the Service SDK for Android:

1. Install the SDK using Gradle.

   The Service SDK is hosted in a maven repository.

Feature	Dependency Name	Maven Repository URL
All Features	com.salesforce.service:servicesdk:224.1.0	https://s3.amazonaws.com/salesforcesos.com/android/maven/release
Knowledge	com.salesforce.service:knowledge-ui:4.3.4 (if you're only using Knowledge Core, then use com.salesforce.service:knowledge-core:4.3.4)	https://s3.amazonaws.com/salesforcesos.com/android/maven/release
Case Management	com.salesforce.service:case-ui:4.2.5 (if you're only using Case Management Core, then use com.salesforce.service:case-core:4.2.5)	https://s3.amazonaws.com/salesforcesos.com/android/maven/release
Chat	com.salesforce.service:chat-ui:4.2.0 (if you're only using Chat Core, then use com.salesforce.service:chat-core:4.2.0)	https://s3.amazonaws.com/salesforcesos.com/android/maven/release
SOS	com.salesforce.service:sos:4.0.9	https://s3.amazonaws.com/salesforcesos.com/android/maven/release, http://tokbox.bintray.com/maven/

To install **all** the Service SDK features, add the following maven repositories to your project's `build.gradle` file.

```
allprojects {
  repositories {
    google()
    jcenter()
    maven {
      url 'https://s3.amazonaws.com/salesforcesos.com/android/maven/release'
    }
    maven {
      url 'http://tokbox.bintray.com/maven/'
    }
  }
}
```

Note: Be sure to put the `repositories` list in the `allprojects` section.

And add the following dependency to your module's `build.gradle` file.

```
dependencies {
  implementation 'com.salesforce.service:servicesdk:224.1.0'
}
```

If you want to use a subset of the SDK features, exclude the undesired dependencies (by using `exclude group` with `knowledge-ui`, `case-ui`, `chat-ui`, or `sos`), or just pick the dependency you want to install. For example, to install **Knowledge** functionality, add the following dependency to your module's `build.gradle` file.

```
dependencies {
  implementation 'com.salesforce.service:knowledge-ui:4.3.4'
}
```

2. Declare permissions.

   To install **Knowledge**, **Case Management**, or **Chat**, you must declare the following permissions in your `AndroidManifest.xml`.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

   To install **SOS** functionality, you must declare the following permissions in your `AndroidManifest.xml`.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"
android:maxSdkVersion="22"/>
```

   Note: If you only support Android Marshmallow (Version 6.0, API level 23) and later, you can remove the `READ_PHONE_STATE` line entirely.

   And if you wish to use **two-way video** with SOS, you must also declare the camera permission.

```
<uses-permission android:name="android.permission.CAMERA"/>
```

You can now start using the Service SDK for Android.

Authentication with the Service SDK for Android

The Service SDK provides an authentication mechanism that allows your users to access user-specific information in Service Cloud. To authenticate, implement two interfaces and provide an access token to the SDK.

`AuthenticatedUser`

`AuthenticatedUser` contains information about the user who wants to be authenticated.

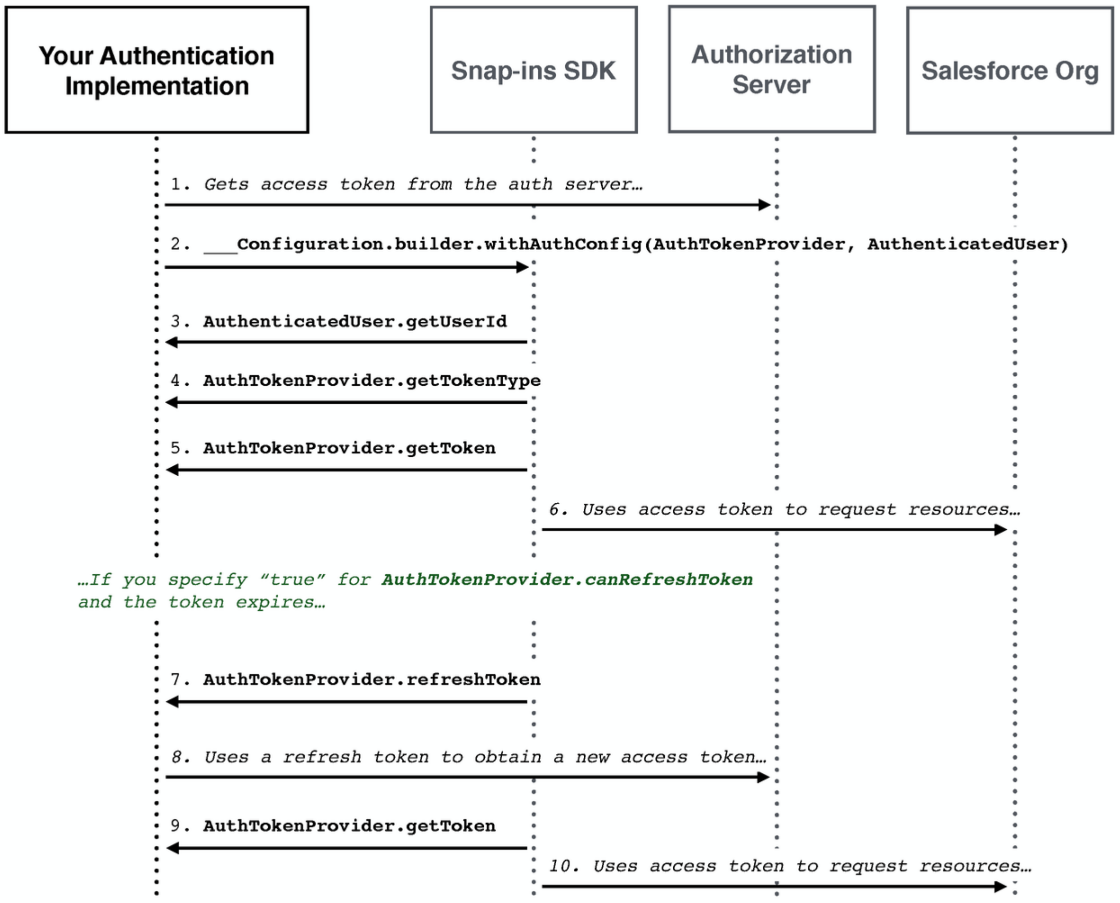
```
public interface AuthenticatedUser {
  @NonNull String getUserId ();
}
```

`AuthTokenProvider`

`AuthTokenProvider` is an interface to the system that obtains the access token. If the access token expires, the Service SDK asks your implementation to refresh the token.

```
public interface AuthTokenProvider {
  @Nullable String getToken ();
  @Nullable String getTokenType ();
  boolean canRefresh ();
  void refreshToken (@NonNull ResponseSummary responseSummary);
}
```

The following sequence diagram describes the basic authentication flow.


Your Authentication Implementation
Snap-ins SDK
Authorization Server
Salesforce Org
1. Gets access token from the auth server…
2. ___Configuration.builder.withAuthConfig(AuthTokenProvider, AuthenticatedUser)
3. AuthenticatedUser.getUserId
4. AuthTokenProvider.getTokenType
5. AuthTokenProvider.getToken
6. Uses access token to request resources…
…If you specify "true" for AuthTokenProvider.canRefreshToken and the token expires…
7. AuthTokenProvider.refreshToken
8. Uses a refresh token to obtain a new access token…
9. AuthTokenProvider.getToken
10. Uses access token to request resources…

Note: In step 2, the configuration class that you call is either `KnowledgeConfiguration` for Knowledge, or `CaseConfiguration` for Case Management. The builder of these two configuration classes contains a `withAuthConfig` method. This method takes your implementation of `AuthTokenProvider` and `AuthenticatedUser`.

If you're using the Salesforce Mobile SDK, you can implement these classes as wrappers to existing authentication features. See Authenticating Using the Salesforce Mobile SDK for more information.

If you're not using the Salesforce Mobile SDK, make sure that your implementation can access whatever authorization server you're using to obtain the access token.

Authenticating Using the Salesforce Mobile SDK
These instructions describe how to authenticate the Service SDK using the provided authentication mechanism within the Salesforce Mobile SDK.

Authenticating Using the Salesforce Mobile SDK

These instructions describe how to authenticate the Service SDK using the provided authentication mechanism within the Salesforce Mobile SDK.

Before starting, make sure that you've already:

- Installed the Service SDK. See Install the Service SDK for Android.

- Installed the Salesforce Mobile SDK. If you want to install this SDK using maven, add a dependency on `com.salesforce.mobilesdk:SalesforceSDK:<VERSION_NUMBER>` to your module's `build.gradle` file.
- Created a connected app that allows the SDK to authenticate with your Salesforce Experience Cloud site. See Connected Apps. For an overview of Salesforce authentication from a mobile device, see Understand Security and Authentication.

Note: When creating a connected app, be sure that it has access to the `chatter_api` scope. See Scope Parameter Values.

1. If you're using a Salesforce Experience Cloud site, be sure to configure the login endpoint as described in the Salesforce Mobile SDK documentation (Configure the Login Endpoint).

   The documentation describes how to use the first server listed in your `servers.xml` file as the default login location. For example:

```
<servers>
  <server name="Site Login" url="https://MY_SITE_URL.com"/>
</servers>
```

2. Add an `account_type` property to `strings.xml`. The property must be unique to your app to prevent conflicts with other apps that use the Service SDK or the Salesforce Mobile SDK.

```
<string name="account_type">com.mycompany.myapp</string>
```

3. Implement `AuthenticatedUser` as a wrapper to your Mobile SDK user.

   In Java:

```
import com.salesforce.androidsdk.accounts.UserAccount;
import com.salesforce.android.service.common.http.AuthenticatedUser;

public class MobileSdkUser implements AuthenticatedUser {
  private final String mUserId;

  public MobileSdkUser (UserAccount userAccount) {
    mUserId = userAccount.getUserId();
  }

  @Override public String getUserId () {
    return mUserId;
  }
}
```

   In Kotlin:

```
class MobileSdkUser(userAccount: UserAccount) : AuthenticatedUser {
  private val mUserId: String

  init {
    mUserId = userAccount.getUserId()
  }

  override fun getUserId(): String {
    return mUserId
  }
}
```

   Note: `AuthenticatedUser.getUserId` must return the Salesforce user ID (`UserAccount.getUserId`) for the Case Management message feed to display correctly.

4. Implement `AuthTokenProvider` as a wrapper to your Mobile SDK authentication system.

   In Java:

```
import android.support.annotation.NonNull;
import android.support.annotation.Nullable;
import com.salesforce.android.service.common.http.AuthTokenProvider;
import com.salesforce.android.service.common.http.ResponseSummary;
import com.salesforce.androidsdk.rest.ClientManager;

public class MobileSdkAuthTokenProvider implements AuthTokenProvider {

  private final ClientManager.AccMgrAuthTokenProvider mTokenProvider;
  private String mAuthToken;

  public MobileSdkAuthTokenProvider (ClientManager.AccMgrAuthTokenProvider tokenProvider,

                                     String initialToken) {
    mTokenProvider = tokenProvider;
    mAuthToken = initialToken;
  }

  @Nullable @Override public String getToken () {
    return mAuthToken;
  }

  @Nullable @Override public String getTokenType () {
    return "Bearer";
  }

  @Override public boolean canRefresh () {
    return true;
  }

  @Override public void refreshToken (@NonNull ResponseSummary responseSummary) {
    mAuthToken = mTokenProvider.getNewAuthToken();
  }
}
```

   In Kotlin:

```
class MobileSdkAuthTokenProvider(
  private val mTokenProvider: ClientManager.AccMgrAuthTokenProvider,
  private var mAuthToken: String?) : AuthTokenProvider {

  override fun getToken(): String? {
    return mAuthToken
  }

  override fun getTokenType(): String? {
    return "Bearer"
  }

  override fun canRefresh(): Boolean {
    return true
  }
  override fun refreshToken(responseSummary: ResponseSummary) {
    mAuthToken = mTokenProvider.getNewAuthToken()
  }
}
```

5. When your app launches, initialize the Salesforce Mobile SDK.

   ```
   // Initialize the Salesforce Mobile SDK
   SalesforceSDKManager.initNative(context, null, MainActivity.class);
   ```

   For more information, see the SalesforceSDKManager documentation in the Mobile SDK Developer's Guide.

6. Write code that allows a user to log in and log out of their org using the Salesforce Mobile SDK. Make sure the user logs in before showing the Service SDK UI.

   For example:

   ```
   public void login (final Activity activity) {
     SalesforceSDKManager.getInstance()
                         .getClientManager()
                         .getRestClient(activity,
                                        new ClientManager.RestClientCallback() {

       @Override public void authenticatedRestClient (RestClient client) {
         // Handle authenticated state activity here
       }
     });
   }

   public void logout (final Activity activity) {
     SalesforceSDKManager.getInstance().logout(activity, true);
   }
   ```

7. When configuring Knowledge (using `KnowledgeConfiguration`) or Case Management (using `CaseConfiguration`), use the `withAuthConfig` method in the builder and pass in your implementation for `AuthenticatedUser` and `AuthTokenProvider`.

   This code sample illustrates how it works with Knowledge:

   ```
   // Create Knowledge configuration builder
   KnowledgeConfiguration.Builder builder =
     KnowledgeConfiguration.builder(siteUrl).offlineResourceConfig(offlineConfig);

   // Grab a user account from the Salesforce Mobile SDK
   UserAccount userAccount =
     SalesforceSDKManager.getInstance().getUserAccountManager().getCurrentUser();

   if (userAccount != null) {

     // Create an authorization token provider from the Salesforce Mobile SDK
     ClientManager.AccMgrAuthTokenProvider authTokenProvider =
       new ClientManager.AccMgrAuthTokenProvider(
         SalesforceSDKManager.getInstance().getClientManager(),
         userAccount.getInstanceServer(),
         userAccount.getAuthToken(),
   ```

```
      userAccount.getRefreshToken());

  // Create a wrapper with the Salesforce Mobile SDK token provider
  AuthTokenProvider provider =
    new MobileSdkAuthTokenProvider(authTokenProvider,
      userAccount.getAuthToken());

  // Build a config using the token provider and the authenticated user
  builder.withAuthConfig(provider, new MobileSdkUser(userAccount));
}

// And now build a configuration object!
KnowledgeConfiguration config = builder.build();
```

This code sample illustrates how it works with Case Management:

```
// Create Case Management configuration builder
CaseConfiguration.Builder builder =
  new CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)

// Grab a user account from the Salesforce Mobile SDK
UserAccount userAccount =
  SalesforceSDKManager.getInstance().getUserAccountManager().getCurrentUser();

if (userAccount != null) {

  // Create an authorization token provider from the Salesforce Mobile SDK
  ClientManager.AccMgrAuthTokenProvider authTokenProvider =
    new ClientManager.AccMgrAuthTokenProvider(
      SalesforceSDKManager.getInstance().getClientManager(),
      userAccount.getInstanceServer(),
      userAccount.getAuthToken(),
      userAccount.getRefreshToken());

  // Create a wrapper with the Salesforce Mobile SDK token provider
  AuthTokenProvider provider =
    new MobileSdkAuthTokenProvider(authTokenProvider,
      userAccount.getAuthToken());

  // Build a config using the token provider and the authenticated user
  builder.withAuthConfiguration(provider, new MobileSdkUser(userAccount))
         .caseListName(CASE_LIST_NAME);
}

// Create a core configuration instance
CaseConfiguration coreConfiguration = builder.build();
```

At this point, you can use the features of the Service SDK as an authenticated user.

Push Notifications with the Service SDK for Android

To take advantage of push notifications from your org to your app, set up an Apex trigger and configure your app for notifications. Pass relevant notification information, such as case feed activity, to the Service SDK using your `PushNotificationListener` implementation.

1. Set up authentication in your app. To learn more, see Authentication with the Service SDK for Android.
2. Add FCM (Firebase Cloud Messaging) to your app. To learn more, see Google's documentation, Add Firebase to Your Android Project.

   Note: Avoid any references to GCM (Google Cloud Messaging) in your build dependencies and your `AndroidManifest.xml` files. If you experience a `NoClassDefError` which claims that `GcmReceiver` is missing, it may be included in your final merged manifest by one of your dependencies, such as the Salesforce Mobile SDK. To resolve the error, add the following remove instruction to your `AndroidManifest.xml`: `<receiver android:name="com.google.android.gms.gcm.GcmReceiver" tools:node="remove"/>`.

3. Add a Service SDK `connected-app` dependency in your module's `build.gradle` file.

```
dependencies {

  // Add the connected-app dependency
  implementation "com.salesforce.service:connected-app:8.0.2"

  // ... your other dependencies go here too ...
}
```

   Note: The version of the connected-app is the same as the version of the Common module used by the Service SDK.

4. Create a connected app and an Apex trigger to send a notification from your org.

   Follow the instructions in the Salesforce Mobile Push Notifications Implementation Guide for Creating a Connected App.

   For a sample Apex trigger for case feed activity, see Push Notifications for Case Activity.

5. Implement `PushNotificationListener` and handle the push notification event in the `onPushNotificationReceived` method.

   For example, the following code extracts the case ID from the payload and passes it to `CaseUIClient`:

   In Java:

```
public class MyPushNotificationListener implements PushNotificationListener {

  @Override public void onPushNotificationReceived (RemoteMessage message) {

    // Extract the Case ID from the data. The name of the key depends
    // on how you bundled the freeform data in your Apex trigger...
    String caseId = message.getData().get("caseid");

    // TO DO: Extract any other info from the data

    CaseUIClient uiClient = // TO DO: Get the UI client from configuration

    if (uiClient != null) {
      // Pass case update information to the UI client.
      uiClient.notifyCaseUpdated(caseId);
    }
  }
}
```

   In Kotlin:

```
class MyPushNotificationListener : PushNotificationListener {
  override fun onPushNotificationReceived (message: RemoteMessage?) {

    // Extract the Case ID from the data. The name of the key depends
    // on how you bundled the freeform data in your Apex trigger...
    val caseId = message.getData().get("caseid")

    // TO DO: Extract any other info from the data

    var uiClient: CaseUIClient? = null

    // TO DO: Get the UI client from configuration

    if (uiClient != null) {
      // Pass case update information to the UI client.
      uiClient.notifyCaseUpdated(caseId)
    }
  }
}
```

6. Configure `SalesforceConnectedApp` using the Sender ID, `AuthTokenProvider`, and the site URL.

   Pass this object your push notification listener and then register your device for push notifications.

   In Java:

```
// Create a connected app object
SalesforceConnectedApp connectedApp =
  SalesforceConnectedApp.create(this, new ConnectedAppConfiguration.Builder()
    .gcmSenderId(uniqueProjectId)
    .salesforceInstanceURL(siteUrl)
    .authTokenProvider(authProvider)
    .build());

// Add the push notification listener
connectedApp.addPushNotificationListener(myPushNotificationListener);

// Register for push notifications
connectedApp.registerDeviceForPushNotifications().onError(new Async.ErrorHandler() {
  @Override public void handleError (Async<?> operation, @NonNull Throwable throwable)
 {
    // TO DO: Handle error
  }
});
```

   In Kotlin:

```
// Create a connected app object
val connectedApp = SalesforceConnectedApp.create(this, ConnectedAppConfiguration.Builder()

    .gcmSenderId(uniqueProjectId)
    .salesforceInstanceURL(siteUrl)
    .authTokenProvider(authProvider)
    .build())

// Add the push notification listener
connectedApp.addPushNotificationListener(myPushNotificationListener)
```

```
// Register for push notifications
connectedApp.registerDeviceForPushNotifications().onError(object: Async.ErrorHandler {
  override fun handleError(operation: Async<*>?, throwable: Throwable) {
    // TO DO: Handle error
  }
})
```

Analytics with the Service SDK for Android

You can listen to user-driven events from the Service SDK using the `ServiceAnalytics` system.

Implement `ServiceAnalyticsListener` and add your listener to `ServiceAnalytics` to start receiving events.

In Java:

```
ServiceAnalytics.addListener(new ServiceAnalyticsListener() {
  @Override public void onServiceAnalyticsEvent(String behaviorId,
                                                Map<String, Object> eventData) {
    // TO DO: Do something with analytics data
  }
});
```

In Kotlin:

```
ServiceAnalytics.addListener { behaviorId, eventData ->
  // TO DO: Do something with analytics data
}
```

When you receive an event, inspect the `behaviorId` to see the behavior that caused the event (for example, `KNOWLEDGE_UI_USER_LAUNCH`). Inspect the `eventData` map for contextual data related to the event (for example, `KnowledgeUIAnalytics.DATA_CATEGORY_GROUP_NAME`).

For a list of behaviors and the key constants for parsing the `eventData` map, see the analytics class for the desired feature.

Knowledge Analytics
`KnowledgeUIAnalytics`

Case Management Analytics
`CasesUIAnalytics`

Chat Analytics
`ChatAnalytics`

Decrease the Size of Your App

Although the SDK doesn't have a large footprint, you can decrease the size of your app by splitting your APK or by using ProGuard.

The SOS SDK contains native libraries for a variety of architectures, so it can benefit from APK splitting. See the Android documentation on splitting your APK.

You can also use ProGuard to shrink and optimize your app. Use the following ProGuard rules as a starting point.

Warning: When stripping code with ProGuard, be sure to use these rules so that you don't unintentionally remove code that is critical to the functioning of the SOS SDK.

```
################# ALL #################

# ------------------ OkHttp --------------------------
-dontwarn okio.**
-dontwarn okhttp3.**
-dontwarn javax.annotation.**
-dontwarn org.conscrypt.**
-keepnames class okhttp3.internal.publicsuffix.PublicSuffixDatabase

# ------------------ SQLCipher ------------------------
# If you're only using Chat, you can remove the sqlcipher rules
-keep class net.sqlcipher.** { *; }
-dontwarn net.sqlcipher.**

# ------------------ Gson ----------------------------
-keepclassmembers,allowobfuscation class * {
    @com.google.gson.annotations.SerializedName <fields>;
}

############### SOS ONLY ###############
# You can delete this section if you are not using SOS

-dontwarn lombok.**

# Our components are initialized using reflection and can appear to be unused
-keepclassmembers class * implements com.salesforce.android.sos.component.Component

# ------------------ Eventbus SOS ------------------------
# The onEvent methods are called from the EventBus library and can appear unused.
-keepclassmembers class com.salesforce.android.sos.** {
    public void onEvent(...);
}

# ------------------ Opentok SOS -------------------------
# OpenTok cannot handle any code stripping for optimization.
-keep class com.opentok.** { *; }
-keep class org.webrtc.** { *; }
-keep class com.salesforce.android.sos.** { *; }

# ------------------ Gson SOS ----------------------------
# Preserve the special static methods that are required in all enumeration classes.
# We use these predominantly for serializing enums with Gson.
-keepclassmembers enum com.salesforce.android.sos.** {
    **[] $VALUES;
    public *;
}
```

Android Examples

Use these examples to learn more about the Service SDK.

Check out our sample apps on GitHub (github.com/forcedotcom/ServiceSDK-Android).

Using Chat with the Service SDK

Add the Chat experience to your mobile app.

Chat in the Service SDK for Android

Using Chat within the Service SDK, you can provide real-time chat sessions from within your native app.

Quick Setup: Chat in the Service SDK

To add Chat to your Android app, create a configuration object that points to your org and then create a Chat UI client.

Use Einstein Bots with Chat

With Einstein Bots, you can complement your chat support experience with a smart, automated system that saves your agents time and keeps your customers happy. Once you've set up Einstein Bots in your org, the SDK automatically begins the chat experience using your bot. You can design your bot to transfer to an agent at any point.

Display Knowledge Article Previews in Chat

By implementing `ChatKnowledgeArticlePreviewDataProvider` and `ChatKnowledgeArticlePreviewClickListener`, you can display Knowledge article previews in a chat session and then show the article details when the user taps on the preview.

Listen for State Changes and Events

You can add listeners for state changes and events during a chat session and respond accordingly. For instance, when the client ends a session, you can display a dialog to the user.

Show Pre-Chat Fields to User

Before a chat session begins, you can request that the user enter pre-chat fields that are sent to the agent at the start of the session.

Create or Update Salesforce Records from a Chat Session

When a chat session begins, you can create or find records within your org and pass this information to the agent. Using this technique, your agent can immediately have all the context they need for an effective chat session.

Check Agent Availability

Before starting a session, you can check the availability of your chat agents and then provide your users with more accurate expectations. For instance, when no agents are available, you can hide or disable the button to contact an agent

Transfer File to Agent

Give users the ability to transfer files during a chat so they can share information about their issues.

Block Sensitive Data in a Chat Session

To block sending sensitive data to agents, specify a regular expression in your org's setup. When the regular expression matches text in the user's message, the matched text is replaced with customizable text before it leaves the device.

Build Your Own UI with the Chat Core API

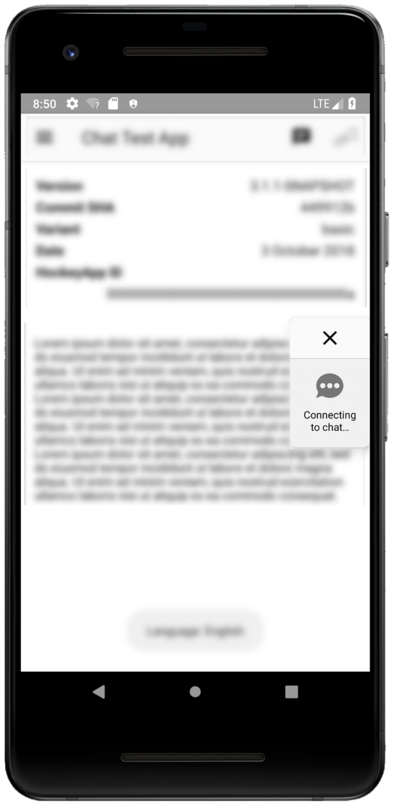
With the Chat Core API, you can access the functionality of Chat without a UI. This API is useful if you want to build your own UI and not use the default.

Chat in the Service SDK for Android

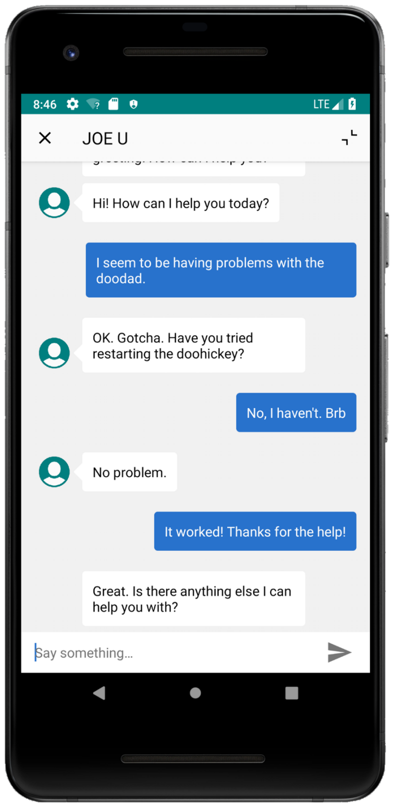
Using Chat within the Service SDK, you can provide real-time chat sessions from within your native app.

Once you've set up chat for Service Cloud, it takes just a few calls to the SDK to have your app ready to handle agent chat sessions.

When a chat session initiates, it is minimized by default so the user can keep using the app.



When the user taps the chat thumbnail, the app goes full screen.



Hi! How can I help you today?

I seem to be having problems with the doodad.

OK. Gotcha. Have you tried restarting the doohickey?

No, I haven't. Brb

No problem.

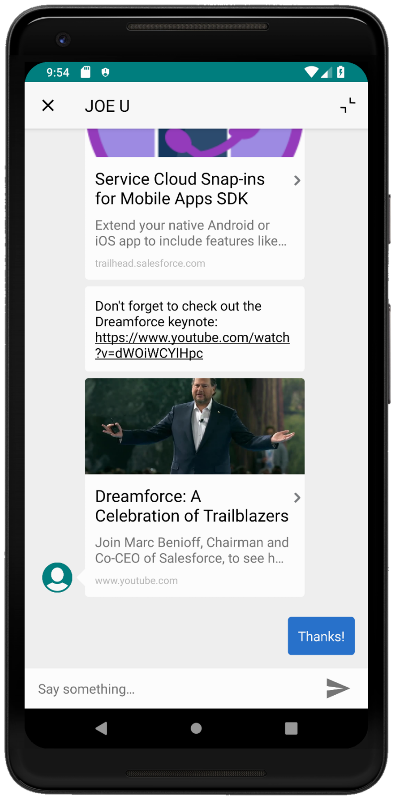
It worked! Thanks for the help!

Great. Is there anything else I can help you with?

If you've implemented Einstein Bots, the user first speaks with a chat bot on page 55 before optionally getting transferred to an agent.

If the user isn't using the app during a chat session, they receive a new message notification when an agent sends a message.

When an agent sends links to your users, they see link previews right from within the chat session. The SDK tries to use Open Graph meta tags (`og:title`, `og:description`, `og:image`) to extract relevant information for the preview.


9:54
JOE U
Service Cloud Snap-ins for Mobile Apps SDK
Extend your native Android or iOS app to include features like...
trailhead.salesforce.com
Don't forget to check out the Dreamforce keynote: https://www.youtube.com/watch?v=dWOiWCYlHpc
Dreamforce: A Celebration of Trailblazers
Join Marc Benioff, Chairman and Co-CEO of Salesforce, to see h...
www.youtube.com
Thanks!
Say something...

You can also customize the look and feel of the interface so that it fits naturally within your app.

Quick Setup: Chat in the Service SDK

To add Chat to your Android app, create a configuration object that points to your org and then create a Chat UI client.

Before starting, make sure that you've already:

- Set up your console to work with Chat. See Org Setup for Chat in Lightning Experience with a Guided Flow.
- Installed the SDK. See Install the Service SDK for Android.

To set up Chat with the default UI, create a configuration object that points to your org and then start a chat session. If you prefer to build your own user interface, see Build Your Own UI with the Chat Core API.

1. Specify your Chat org settings.

   In Java:

```
public static final String ORG_ID = "YOUR_ORG_ID";
public static final String DEPLOYMENT_ID = "YOUR_DEPLOYMENT_ID";
public static final String BUTTON_ID = "YOUR_BUTTON_ID";
public static final String LIVE_AGENT_POD = "YOUR_LAC_ORG_URL";
                                    // e.g. "d.la.salesforce.com"
```

In Kotlin:

```
val ORG_ID = "YOUR_ORG_ID"
val DEPLOYMENT_ID = "YOUR_DEPLOYMENT_ID"
val BUTTON_ID = "YOUR_BUTTON_ID"
val LIVE_AGENT_POD = "YOUR_LAC_ORG_URL"
             // e.g. "d.la.salesforce.com"
```

Note: You can get the required parameters from your Salesforce org. If your Salesforce admin hasn't set up Chat in Service Cloud or you need more guidance, see Org Setup for Chat in Lightning Experience with a Guided Flow.

2. Create a `ChatConfiguration` object using your org settings.

   In Java:

```
// Create a core configuration instance
ChatConfiguration chatConfiguration =
  new ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                                DEPLOYMENT_ID, LIVE_AGENT_POD)
                                .build();
```

   In Kotlin:

```
// Create a core configuration instance
val chatConfiguration =
  ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                            DEPLOYMENT_ID, LIVE_AGENT_POD)
                            .build()
```

   Tip: You can use the `ChatConfiguration.Builder` class to configure other behaviors, such as how to handle the pre-chat view (see Show Pre-Chat Fields to User) and how to specify the name of the visitor speaking with the agent (use the `visitorName` method).

3. Configure and launch a chat session using `ChatUI`, `ChatUIConfiguration`, and `ChatUIClient`.

   In Java:

```
// Create a UI configuration instance from a core config object
// and start session!
ChatUI.configure(ChatUIConfiguration.create(chatConfiguration))
  .createClient(getApplicationContext())
  .onResult(new Async.ResultHandler<ChatUIClient>() {
      @Override public void handleResult (Async<?> operation,
        ChatUIClient chatUIClient) {
            chatUIClient.startChatSession(MainActivity.this);
      }
});
```

   In Kotlin:

```
// Create a UI configuration instance from a core config object
// and start session!
ChatUI.configure(ChatUIConfiguration.create(chatConfiguration))
  .createClient(applicationContext)
  .onResult { _,
    chatUIClient -> chatUIClient.startChatSession(this@MainActivity)
  }
```

Warning: Don't create more than 1 instance of `ChatUIClient`. The system returns an `Async` error if there is already an active instance. The SDK only supports 1 session at a time per device. Wait until the chat session ends before attempting to start a new session.

When calling `startChatSession`, pass the context for the `Activity` that you want to show the chat UI on top of. The chat session starts minimized and the user can tap the thumbnail to go full screen. If you want the session to start in full-screen mode, call `defaultToMinimized(false)` on the builder.

Note: When the minimized view is visible, it displays the number of unread messages. This value represents the total number of bot, agent, and system messages that are unread.

By default, the user's queue position is shown while the user waits for an agent. You can change this information from a position number to an estimated wait time using the `queueStyle` build method and specifying `QueueStyle.EstimatedWaitTime`. When using the estimated wait time, you can set the minimum (`minimumWaitTime`) and maximum (`maximumWaitTime`) wait time values. If the wait time exceeds the maximum value, a generic message appears, which you can customize on page 161 (using the customizable chat strings). To understand the algorithm used for the estimated wait time, see the estimated wait time documentation in the Chat REST API Developer Guide. To hide queue information entirely, use a queue style of `None`.

Sample alternate configuration in Java:

```
ChatUIConfiguration uiConfig = new ChatUIConfiguration.Builder()
  .chatConfiguration(chatConfiguration)
  .queueStyle(QueueStyle.EstimatedWaitTime) // Use estimated wait time
  .defaultToMinimized(false)                // Start in full-screen mode
  .build();
```

Sample alternate configuration in Kotlin:

```
val uiConfig = ChatUIConfiguration.Builder()
  .chatConfiguration(chatConfiguration)
  .queueStyle(QueueStyle.EstimatedWaitTime) // Use estimated wait time
  .defaultToMinimized(false)                // Start in full-screen mode
  .build()
```

4. (Optional) Customize the interface.

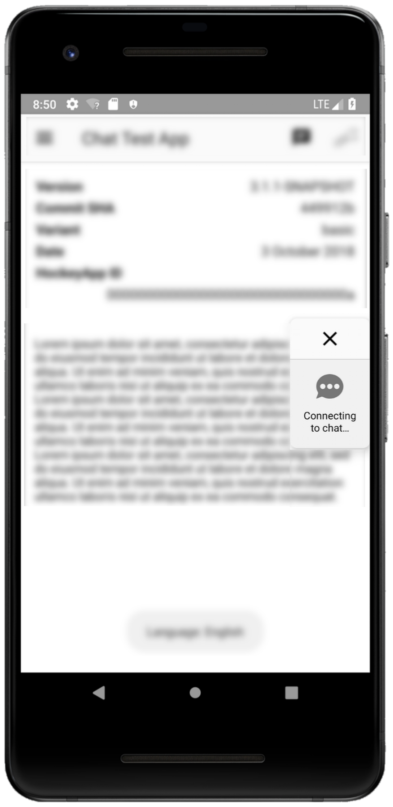
You can customize the colors, strings, and other aspects of the interface. You can also localize the strings into other languages.

5. (Optional) Add listeners for state changes or events.

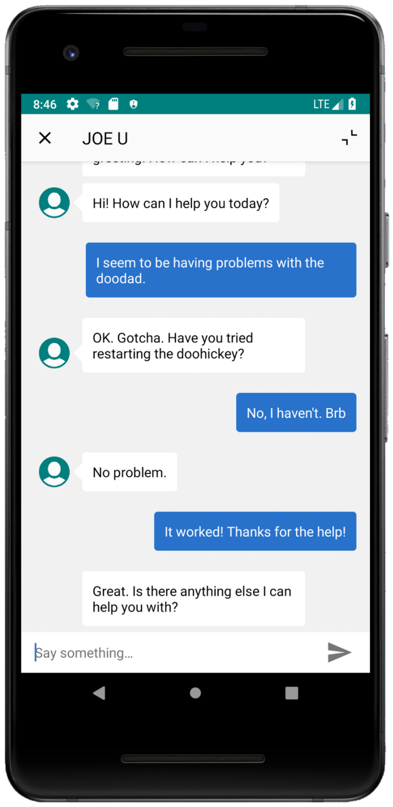
You can add listeners for state changes and events during a chat session and respond accordingly. For instance, when the client ends a session, you can display a dialog to the user. See Listen for State Changes and Events.

Warning: If you use an incorrect button ID in your `ChatConfiguration`, the chat fails and your `SessionStateListener` reports a `ChatEndReason` of `NetworkError`.

When a session launches, it appears minimized.



The user can tap the session to make it full screen and begin a conversation with an agent.



Hi! How can I help you today?

I seem to be having problems with the doodad.

OK. Gotcha. Have you tried restarting the doohickey?

No, I haven't. Brb

No problem.

It worked! Thanks for the help!

Great. Is there anything else I can help you with?

Say something...

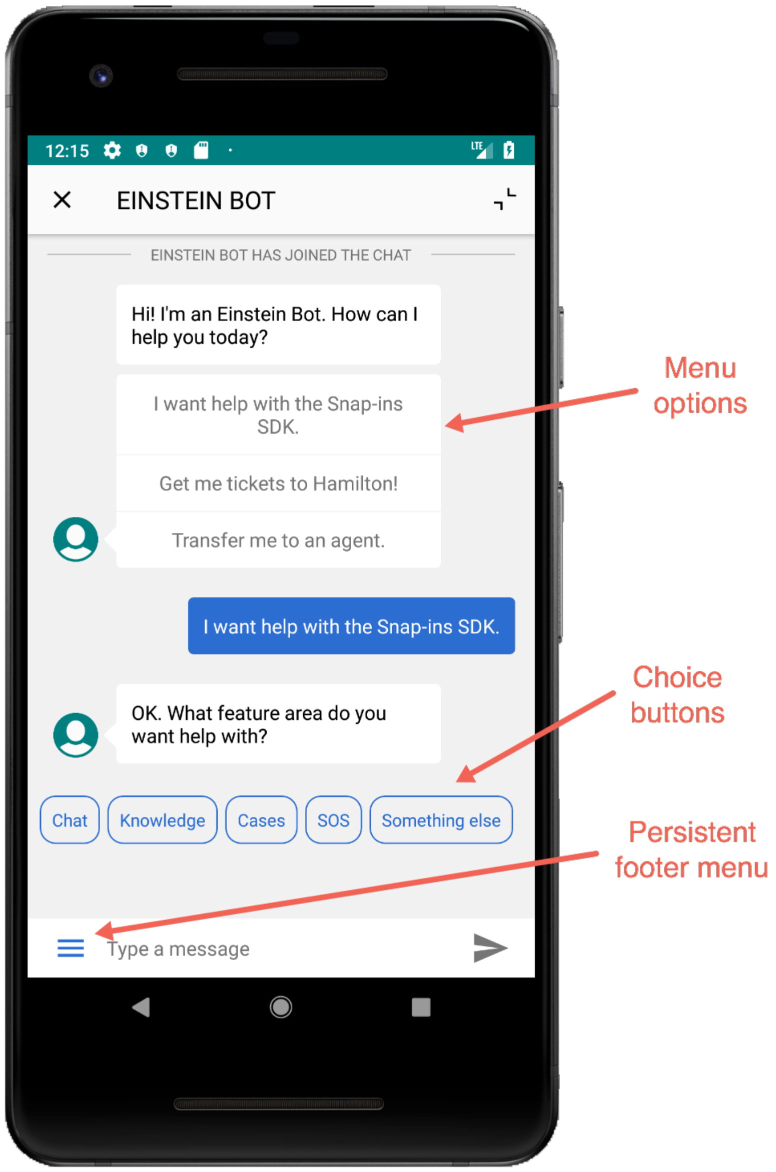
Note: When the app is in the background, the SDK ensures that the session remains open. A timeout should only occur when there is a connection issue, the agent closes the session, or the app is removed from memory.

Use Einstein Bots with Chat

With Einstein Bots, you can complement your chat support experience with a smart, automated system that saves your agents time and keeps your customers happy. Once you've set up Einstein Bots in your org, the SDK automatically begins the chat experience using your bot. You can design your bot to transfer to an agent at any point.

Before you can use Einstein Bots in your mobile app, enable and build a bot in your org. To learn more, see Einstein Bots in Salesforce Help. In broad strokes, you must enable Einstein Bots, deploy the bot to your channel, and activate the bot. If you want to learn about building a more robust bot, see the Einstein Bots Developer Cookbook.

Once you've set up your bot and assigned it to your chat button, a chat session automatically starts out as a bot. The menu options, choice buttons, and persistent footer menu that you designed for your bot all appear from within the mobile chat session. These features give your customers direct ways to get what they need—fast.



Hi! I'm an Einstein Bot. How can I help you today?

I want help with the Snap-ins SDK.

Get me tickets to Hamilton!

Transfer me to an agent.

I want help with the Snap-ins SDK.

OK. What feature area do you want help with?

You do have a few ways you can fine-tune the bot from the SDK.

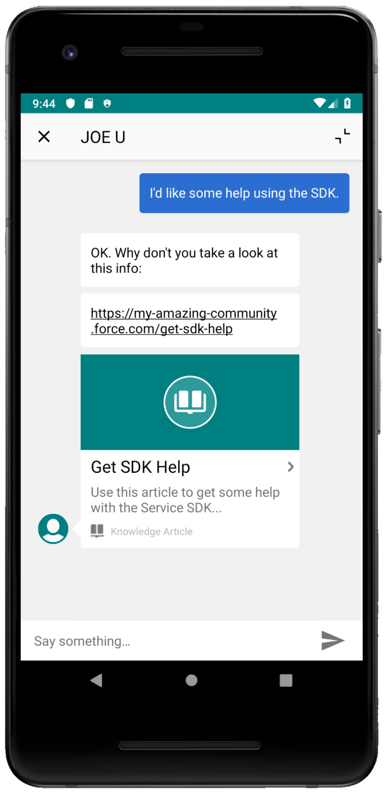
Feature Area	Details

Einstein Bot Avatar	Configure the bot avatar that displays during a session with a bot. To do this, add your `Drawable` to the `ChatUIConfiguration` builder using the `chatBotAvatar` method.
	`final ChatUIConfiguration.Builder uiConfigBuilder =`
	`  new ChatUIConfiguration.Builder();`
	`uiConfigBuilder.chatBotAvatar(R.drawable.my_chatbot_avatar);`
	`uiConfigBuilder.build();`

Einstein Bot Banner	Configure the banner that displays during a session with a bot. To do this, add your Layout to the `ChatUIConfiguration` builder using the `enableChatBotBanner` method.
	`final ChatUIConfiguration.Builder uiConfigBuilder =`
	`  new ChatUIConfiguration.Builder();`
	`uiConfigBuilder.enableChatBotBanner(R.layout.my_chatbot_banner);`
	`uiConfigBuilder.build();`

Display Knowledge Article Previews in Chat

By implementing `ChatKnowledgeArticlePreviewDataProvider` and `ChatKnowledgeArticlePreviewClickListener`, you can display Knowledge article previews in a chat session and then show the article details when the user taps on the preview.


9:44
JOE U
I'd like some help using the SDK.
OK. Why don't you take a look at this info:
https://my-amazing-community.force.com/get-sdk-help
Get SDK Help
Use this article to get some help with the Service SDK...
Knowledge Article
Say something...

Before implementing this feature, you should be familiar with creating a `KnowledgeClient` instance and fetching articles. To learn more, see Article Fetching with the Knowledge Core API.

1. Create a `ChatKnowledgeArticlePreviewDataProvider` implementation to get data about knowledge articles. Within the `onPreviewDataRequested` method, call the `ChatKnowledgeArticlePreviewDataHelper` object with the article title and summary.

```
public class ArticlePreviewDataProvider implements ChatKnowledgeArticlePreviewDataProvider
 {
  private KnowledgeClient mKnowledgeClient;

  ArticlePreviewDataProvider (@NonNull KnowledgeClient knowledgeClient) {
    mKnowledgeClient = knowledgeClient;
  }

  @Override
  public boolean onPreviewDataRequested(final String articleIdOrTitle,
    final ChatKnowledgeArticlePreviewDataHelper helper) {

    // Request info about the Knowledge article
    ArticleDetailRequest articleDetailRequest =
      ArticleDetailRequest.builder(articleIdOrTitle).build();
    mKnowledgeClient.submit(articleDetailRequest)
      .onResult(new Async.ResultHandler<ArticleDetails>() {

      @Override
      public void handleResult(Async<?> operation, @NonNull ArticleDetails result) {

        // Pass resulting information about article to the data helper object
        helper.onPreviewDataReceived(result.getTitle(), result.getSummary());
      }
    }).onComplete(new Async.CompletionHandler() {
      @Override
      public void handleComplete(Async<?> operation) {
        // Handle completion
      }
    }).onError(new Async.ErrorHandler() {
      @Override
      public void handleError(Async<?> operation, @NonNull Throwable throwable) {
        // Pass null info to the data helper object when there is an error
        helper.onPreviewDataReceived(null, null);
      }
    });
    return true;
  }
}
```

2. Create a `ChatKnowledgeArticlePreviewClickListener` implementation to handle when the user taps the link. Within the `onClick` method, show the article detail view.

```
public class ArticlePreviewClickListener
  implements ChatKnowledgeArticlePreviewClickListener {

  private KnowledgeUIClient mKnowledgeUIClient;

  ArticlePreviewClickListener (@NonNull KnowledgeUIClient knowledgeUIClient) {
    mKnowledgeUIClient = knowledgeUIClient;
  }

  @Override
  public boolean onClick (final Context context, final String articleIdOrTitle) {

    // Request info about the Knowledge article
    ArticleDetailRequest articleDetailRequest =
      ArticleDetailRequest.builder(articleIdOrTitle).build();
    mKnowledgeUIClient.getKnowledgeCoreClient().submit(articleDetailRequest)
      .onResult(new Async.ResultHandler<ArticleDetails>() {

      @Override
      public void handleResult(Async<?> operation, @NonNull ArticleDetails result) {

        // Launch the article once we've got the result
        mKnowledgeUIClient.launchArticle((Activity) context, result);
      }
    }).onComplete(new Async.CompletionHandler() {
      @Override
      public void handleComplete(Async<?> operation) {
        // Handle completion
      }
    }).onError(new Async.ErrorHandler() {
      @Override
      public void handleError(Async<?> operation, @NonNull Throwable throwable) {
        // TO DO: Handle error
      }
    });
    return true;
  }
}
```

3. When you create the `ChatUIConfiguration`, pass along enough information to support knowledge article previews.

   Use the `knowledgeCommunityUrl` method to pass info about the Experience Cloud site URL; use the `knowledgeArticlePreviewDataProvider` method to pass the data provider implementation; use the `knowledgeArticlePreviewClickListener` method to pass the click listener implementation.

```
ChatUIConfiguration.Builder uiConfigurationBuilder = chatUIConfigurationBuilder
  .chatConfiguration(chatConfigurationBuilder.build())
  .enableHyperlinkPreview(true)
  .knowledgeCommunityUrl(communityUrl)
  .knowledgeArticlePreviewDataProvider(
      new ArticlePreviewDataProvider(mKnowledgeClient))
  .knowledgeArticlePreviewClickListener(
      new ArticlePreviewClickListener(mKnowledgeUIClient));
```

Listen for State Changes and Events

You can add listeners for state changes and events during a chat session and respond accordingly. For instance, when the client ends a session, you can display a dialog to the user.

1. Create a `SessionStateListener` implementation to handle session state changes.

   In Java:

```
public class MySessionStateListener implements SessionStateListener {

  @Override public void onSessionStateChange (ChatSessionState state) {
    if (state == ChatSessionState.Disconnected) {
      // TODO: Handle the disconnected state change
    }
  }

  @Override public void onSessionEnded (ChatEndReason endReason) {
    if (endReason == ChatEndReason.EndedByAgent) {
      // TODO: Show a UI telling the user that the agent ended the session
    }
  }
}
```

   In Kotlin:

```
class MySessionStateListener: SessionStateListener {

  override fun onSessionStateChange(state: ChatSessionState?) {
    if (state == ChatSessionState.Disconnected) {
      // TODO: Handle the disconnected state change
    }
  }
  override fun onSessionEnded(endReason: ChatEndReason?) {
    if (endReason == ChatEndReason.EndedByAgent) {
      // TODO: Show a UI telling the user that the agent ended the session
    }
  }
}
```

This implementation handles state changes (`onSessionStateChange`) and why the session ended (`onSessionEnded`). For information on other session states and reasons for ending, see `ChatSessionState` and `ChatEndReason`.

2. Create a `ChatEventListener` implementation if you want to listen for additional events.

   This listener isn't required, but it can be used to listen for events such as: when an agent joins (`agentJoined`), when a message is sent (`processedOutgoingMessage`), when a message is received (`didReceiveMessage`).

   In Java:

```
public class MyEventListener implements ChatEventListener {
    public void agentJoined (AgentInformation agentInformation) {
        // Handle agent joined
    }

    public void processedOutgoingMessage (String message) {
        // Handle outgoing message processed
    }

    public void didSelectMenuItem (ChatWindowMenu.MenuItem menuItem) {
        // Handle chatbot menu selected
    }

    public void didSelectButtonItem (ChatWindowButtonMenu.Button buttonItem) {
        // Handle chatbot button selected
    }

    public void didSelectFooterMenuItem (ChatFooterMenu.MenuItem footerMenuItem) {
        // Handle chatboot footer menu selected
    }

    public void didReceiveMessage (ChatMessage chatMessage) {
        // Handle received message
    }

    public void transferToButtonInitiated () {
        // Handle transfer to agent
    }

    public void agentIsTyping (boolean isUserTyping) {
        // Handle typing update
    }
}
```

   In Kotlin:

```
class MyEventListener : ChatEventListener {
    override fun agentJoined(agentInformation: AgentInformation) {
        // Handle agent joined
    }

    override fun processedOutgoingMessage(message: String) {
        // Handle outgoing message processed
    }

    override fun didSelectMenuItem(menuItem: ChatWindowMenu.MenuItem) {
        // Handle chatbot menu selected
    }

    override fun didSelectButtonItem(buttonItem: ChatWindowButtonMenu.Button) {
        // Handle chatbot button selected
    }

    override fun didSelectFooterMenuItem(footerMenuItem: ChatFooterMenu.MenuItem) {
        // Handle chatboot footer menu selected
    }

    override fun didReceiveMessage(chatMessage: ChatMessage) {
        // Handle received message
    }

    override fun transferToButtonInitiated() {
        // Handle transfer to agent
    }

    override fun agentIsTyping(isUserTyping: Boolean) {
        // Handle typing update
    }
}
```

3. Create a `SessionInfoListener` implementation if you want to get session information.

   In Java:

```
public class MySessionInfoListener implements SessionInfoListener {

    public void onSessionInfoReceived (ChatSessionInfo chatSessionInfo) {
        // TO DO: Do something with the session ID
        String sessionId = chatSessionInfo.getSessionId();
}
```

   In Kotlin:

```
class MySessionInfoListener : SessionInfoListener {

    override fun onSessionInfoReceived(chatSessionInfo: ChatSessionInfo) {
        // TO DO: Do something with the session ID
        val sessionId = chatSessionInfo.getSessionId()
    }
}
```

4. Instantiate your listener instances from your `Application` class.

In Java:

```java
private MySessionStateListener mSessionStateListener;
private MyEventListener mEventListener;
private MySessionInfoListener mSessionInfoListener;

@Override public void onCreate () {
  super.onCreate();
  mSessionStateListener = new MySessionStateListener();
  mEventListener = new MyEventListener();
  mSessionInfoListener = new MySessionInfoListener();
}

public SessionStateListener getSessionStateListener () {
  return mSessionStateListener;
}

public ChatEventListener getEventListener () {
  return mEventListener;
}

public SessionInfoListener getSessionInfoListener () {
  return mSessionInfoListener;
}
```

In Kotlin:

```kotlin
val mSessionStateListener = MySessionStateListener()
val mEventListener = MyEventListener()
val mSessionInfoListener = MySessionInfoListener()
```

Note: It's important to have the listener at the `Application` scope to ensure that the session is trackable throughout the lifetime of the application rather than just within an `Activity`.

5. When you configure and start a session, add the listeners that you implemented. Add the event listener (`ChatEventListener`) to the chat UI configuration object. Add the session info listener (`SessionInfoListener`) and the session state listener (`SessionStateListener`) to the chat UI client.

   In Java:

```java
// Create a UI configuration instance from a core instance
// and add our event listener
ChatUIConfiguration.Builder uiConfig = new ChatUIConfiguration.Builder()
  .chatConfiguration(chatConfiguration)
  .chatEventListener((MyApplication)getApplication()).getEventListener());

// Create a chat session and add our session listener
ChatUI.configure(uiConfig.build())
  .createClient(getApplicationContext())
  .onResult(new Async.ResultHandler<ChatUIClient>() {
      @Override public void handleResult (Async<?> operation,
        ChatUIClient chatUIClient) {

            SessionStateListener sessionStateListener =
                ((MyApplication)getApplication()).getSessionStateListener();
            chatUIClient.addSessionStateListener(sessionStateListener);
            SessionInfoListener sessionInfoListener =
                ((MyApplication)getApplication()).getSessionInfoListener();
            chatUIClient.addSessionInfoListener(sessionInfoListener);

            chatUIClient.startChatSession(MainActivity.this);
        }
});
```

In Kotlin:

```
// Create a UI configuration instance from a core instance
// and add our event listener
val uiConfig = ChatUIConfiguration.Builder()
    .chatConfiguration(chatConfiguration)
    .chatEventListener(mEventListener)

// Create a chat session and add our session listener
ChatUI.configure(uiConfig.build())
    .createClient(applicationContext)
    .onResult { operation, chatUIClient ->
        chatUIClient.addSessionStateListener(mSessionStateListener)
        chatUIClient.addSessionInfoListener(mSessionInfoListener)
        chatUIClient.startChatSession(this@MainActivity)
    }
```

Show Pre-Chat Fields to User

Before a chat session begins, you can request that the user enter pre-chat fields that are sent to the agent at the start of the session.

To create pre-chat fields in your app, instantiate `ChatUserData` objects during session configuration and then pass the pre-chat info to your `ChatConfiguration` builder.

1. Create a `ChatUserData`-derived object for each pre-chat field. Use the subclass `PreChatTextInputField` for string fields, the subclass `PreChatPickListField` for picklists, and you can directly use `ChatUserData` for fields that don't require any user input at all. For fields that require user interaction, specify the display label and the label that the agent sees. You can also specify other characteristics, such as whether the field is required and what type of text field it is. When building the input field object, the first string is what the user sees on the device and the second string is what the agent sees in the transcript.

   In Java:

```
// Some simple string fields
PreChatTextInputField firstName = new PreChatTextInputField.Builder()
  .required(true)
  .build("Please enter your first name", "First Name");
  // First string in build() is what the user sees on the device,
  // the second string is what the agent sees in the transcript...
PreChatTextInputField lastName = new PreChatTextInputField.Builder()
  .required(true)
  .build("Please enter your last name", "Last Name");

// An email field
PreChatTextInputField email = new PreChatTextInputField.Builder()
  .required(true)
  .inputType(EditorInfo.TYPE_TEXT_VARIATION_EMAIL_ADDRESS)
  .mapToChatTranscriptFieldName("Email__c")
  .build("Please enter your email", "Email Address");

// A phone number field (that isn't displayed to agent)
PreChatTextInputField phoneNumber = new PreChatTextInputField.Builder()
  .displayedToAgent(false)
  .inputType(InputType.TYPE_CLASS_PHONE)
  .build("Phone number (Agent can't see this)", "Phone");

// A read-only field
PreChatTextInputField subject = new PreChatTextInputField.Builder()
  .readOnly(true)
  .initialValue("Read-only case subject")
  .build("Case Subject", "Subject");

// A long message field
PreChatTextInputField description = new PreChatTextInputField.Builder()
  .inputType(EditorInfo.TYPE_TEXT_VARIATION_LONG_MESSAGE)
  .maxValueLength(200)
  .build("Please describe your problem", "Description");

// A picklist field
PreChatPickListField priority = new PreChatPickListField.Builder()
  .required(true)
  .addOption(new PreChatPickListField.Option("Low Priority", "Low"))
  .addOption(new PreChatPickListField.Option("Medium Priority", "Medium"))
  .addOption(new PreChatPickListField.Option("High Priority", "High"))
  .addOption(new PreChatPickListField.Option("AHHHHHHHH!!!", "Critical"))
  .build("Issue Priority", "Priority");

// You can also create hidden fields that the user doesn't see, but
// still gets passed along to the agent using ChatUserData...
ChatUserData hiddenField = new ChatUserData(
  "Hidden Custom Data",
  "The user doesn't see this information",
  true);
```

In Kotlin:

```
// Some simple string fields
val firstName = PreChatTextInputField.Builder()
    .required(true)
    .build("Please enter your first name", "First Name")
    // First string in build() is what the user sees on the device,
    // the second string is what the agent sees in the transcript...
val lastName = PreChatTextInputField.Builder()
    .required(true)
    .build("Please enter your last name", "Last Name")

// An email field
val email = PreChatTextInputField.Builder()
    .required(true)
    .inputType(EditorInfo.TYPE_TEXT_VARIATION_EMAIL_ADDRESS)
    .mapToChatTranscriptFieldName("Email__c")
    .build("Please enter your email", "Email Address")
```

```
// A phone number field (that isn't displayed to agent)
val phoneNumber = PreChatTextInputField.Builder()
    .displayedToAgent(false)
    .inputType(InputType.TYPE_CLASS_PHONE)
    .build("Phone number (Agent can't see this)", "Phone")

// A read-only field
val subject = PreChatTextInputField.Builder()
    .readOnly(true)
    .initialValue("Read-only case subject")
    .build("Case Subject", "Subject")

// A long message field
val description = PreChatTextInputField.Builder()
    .inputType(EditorInfo.TYPE_TEXT_VARIATION_LONG_MESSAGE)
    .maxValueLength(200)
    .build("Please describe your problem", "Description")

// A picklist field
val priority = PreChatPickListField.Builder()
    .required(true)
    .addOption(PreChatPickListField.Option("Low Priority", "Low"))
    .addOption(PreChatPickListField.Option("Medium Priority", "Medium"))
    .addOption(PreChatPickListField.Option("High Priority", "High"))
    .addOption(PreChatPickListField.Option("AHHHHHHHH!!!", "Critical"))
    .build("Issue Priority", "Priority")

// You can also create hidden fields that the user doesn't see, but
// still gets passed along to the agent using ChatUserData...
val hiddenField = ChatUserData(
    "Hidden Custom Data",
    "The user doesn't see this information",
    true)
```

2. (Optional) Create a list of `ChatEntity` objects to associate pre-chat fields with fields from a record in your org.

   Pre-chat fields are always sent to the agent at the start of the session. But if you want to fill in fields of a particular record, instantiate a `ChatEntity` for each Salesforce object (for example, `Case` or `Contact`) and add a `ChatEntityField` for each field association within that Salesforce object (for example, `Subject` or `LastName`). After you've built your `ChatEntity` objects, pass them to your `ChatConfiguration` builder using the `chatEntities` method.

   To learn more, see Create or Update Salesforce Records from a Chat Session.

3. Pass the pre-chat info to your `ChatConfiguration` builder using the `chatUserData` method.

   In Java:

```
// Create the chat configuration builder
final ChatConfiguration.Builder chatConfigurationBuilder =
  new ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                                DEPLOYMENT_ID, LIVE_AGENT_POD);

// Add user data and entities
chatConfigurationBuilder
  .chatUserData(firstName, lastName, email, priority,
    subject, description, phoneNumber, hiddenField);

// Build the chat configuration object
ChatConfiguration chatConfiguration = chatConfigurationBuilder.build();
```

In Kotlin:

```
// Create the chat configuration builder
val chatConfigurationBuilder =
    ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                              DEPLOYMENT_ID, LIVE_AGENT_POD)

// Add user data and entities
chatConfigurationBuilder
    .chatUserData(firstName, lastName, email, priority,
        subject, description, phoneNumber, hiddenField)

// Build the chat configuration object
val chatConfiguration = chatConfigurationBuilder.build()
```

From here, you can start the chat session normally.

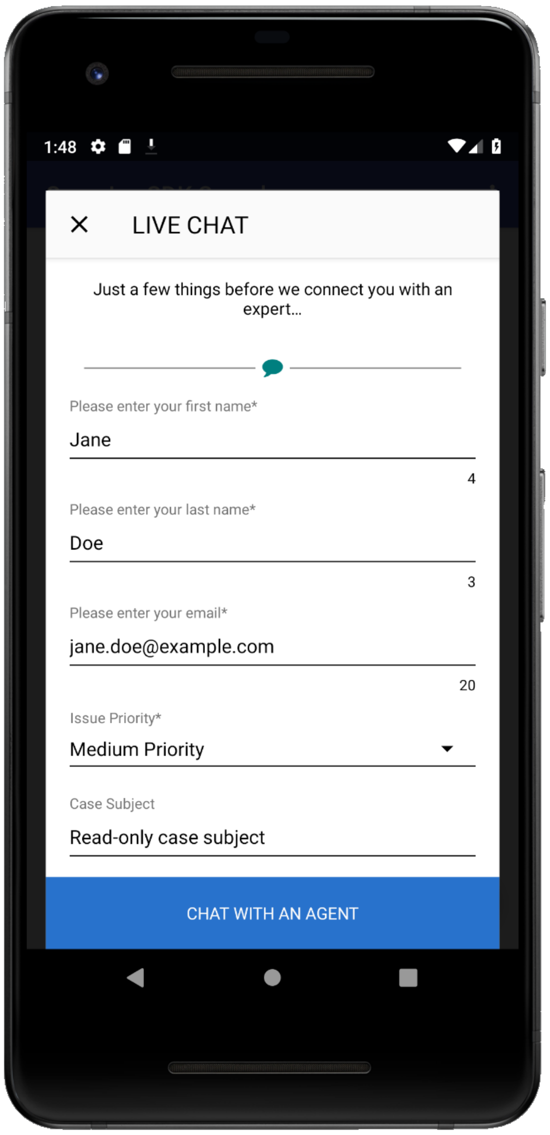
In Java:

```
// Create a UI configuration instance from a core config object
// and start session!
ChatUI.configure(ChatUIConfiguration.create(chatConfiguration))
  .createClient(getApplicationContext())
  .onResult(new Async.ResultHandler<ChatUIClient>() {
      @Override public void handleResult (Async<?> operation,
        ChatUIClient chatUIClient) {
            chatUIClient.startChatSession(MainActivity.this);
      }
});
```

In Kotlin:

```
// Create a UI configuration instance from a core config object
// and start session!
ChatUI.configure(ChatUIConfiguration.create(chatConfiguration))
  .createClient(applicationContext)
  .onResult { _,
    chatUIClient -> chatUIClient.startChatSession(this@MainActivity)
  }
```

With this code, the user sees the following pre-chat UI in their mobile app.



1:48

LIVE CHAT

Just a few things before we connect you with an expert...

Please enter your first name*
Jane
4

Please enter your last name*
Doe
3

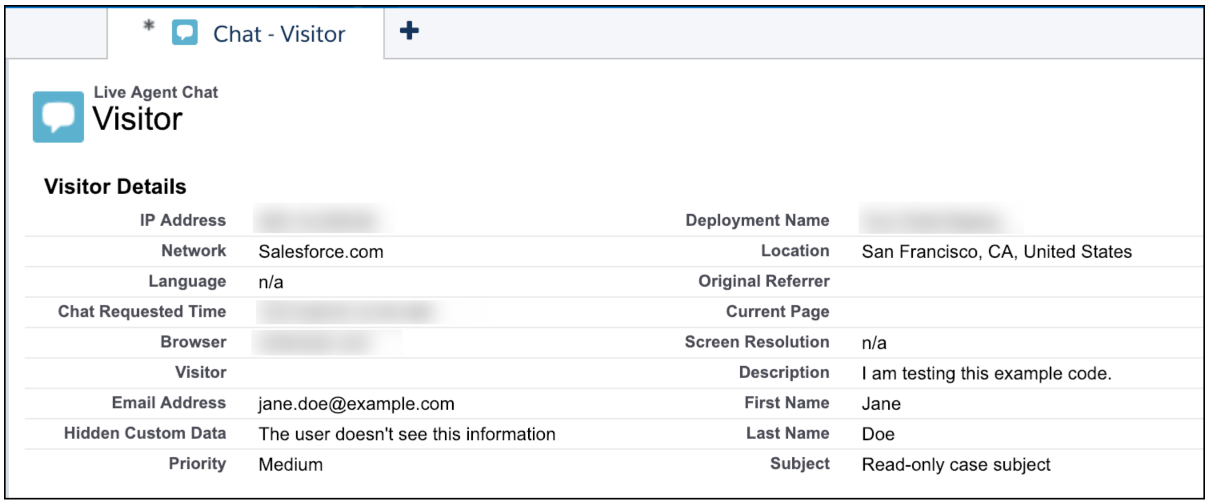
Please enter your email*
jane.doe@example.com
20

Issue Priority*
Medium Priority

Case Subject
Read-only case subject

CHAT WITH AN AGENT

And the agent sees the following UI from the console.



Live Agent Chat
Visitor

Visitor Details

IP Address		Deployment Name	
Network	Salesforce.com	Location	San Francisco, CA, United States
Language	n/a	Original Referrer	
Chat Requested Time		Current Page	
Browser		Screen Resolution	n/a
Visitor		Description	I am testing this example code.
Email Address	jane.doe@example.com	First Name	Jane
Hidden Custom Data	The user doesn't see this information	Last Name	Doe
Priority	Medium	Subject	Read-only case subject

Create or Update Salesforce Records from a Chat Session

When a chat session begins, you can create or find records within your org and pass this information to the agent. Using this technique, your agent can immediately have all the context they need for an effective chat session.

Overview

Before reading these instructions, review Show Pre-Chat Fields to User to understand how to create pre-chat fields.

Pre-chat fields are always sent to the agent at the start of the session. But if you want to fill in fields of a particular record, instantiate a `ChatEntity` for each Salesforce object (for example, `Case` or `Contact`) and add a `ChatEntityField` for each field association within that Salesforce object (for example, `Subject` or `LastName`). After you've built your `ChatEntity` objects, pass them to your `ChatConfiguration` builder using the `chatEntities` method.

Note: Case creation does not currently work for Omni-Channel routing without a setup change to your org. To resolve this problem, raise a ticket with Salesforce to ensure that Omni-Channel is enabled to create a Case in your org.

Basic Flow

This sample shows how to create the first and last name to a contact record in your org. This example doesn't involve user input, but you can use `PreChatTextInputField` instead of `ChatUserData` to allow user input.

In Java:

```
// Create chat user data that doesn't require user interaction
ChatUserData firstNameData = new ChatUserData("FirstName", "Jane", true);
ChatUserData lastNameData = new ChatUserData("LastName", "Doe", true);
```

In Kotlin:

```
// Create chat user data that doesn't require user interacti
val firstNameData = ChatUserData("FirstName", "Jane", true)
val lastNameData = ChatUserData("LastName", "Doe", true)
```

The first argument is the field label that is displayed to the agent in the transcript. The second argument is the value. The third argument is whether this value is displayed to the agent.

After you create your `ChatUserData` objects, create a `ChatEntity` for each Salesforce object that you want to associate these values with. The `ChatEntity` object contains a `ChatEntityField` for each field association. When you build this `ChatEntityField` object, pass in a reference to the associated `ChatUserData` object.

In this example, we create two `ChatEntityField` objects and one `ChatEntity` using those two objects.

In Java:

```
// Build chat entity fields
ChatEntityField firstNameField =
  new ChatEntityField.Builder().doFind(true)
                               .isExactMatch(true)
                               .doCreate(true)
                               .build("FirstName", firstNameData);
ChatEntityField lastNameField =
  new ChatEntityField.Builder().doFind(true)
                               .isExactMatch(true)
                               .doCreate(true)
                               .build("LastName", lastNameData);
```

```
// Build a chat entity object from those fields
// (to map user data to fields in a Salesforce record)
ChatEntity contactEntity = new ChatEntity.Builder()
  .showOnCreate(true)
  .addChatEntityField(firstNameField)
  .addChatEntityField(lastNameField)
  .build("Contact");
```

In Kotlin:

```
// Build chat entity fields
val firstNameField = ChatEntityField.Builder().doFind(true)
                                              .isExactMatch(true)
                                              .doCreate(true)
                                              .build("FirstName", firstNameData)
val lastNameField = ChatEntityField.Builder().doFind(true)
                                             .isExactMatch(true)
                                             .doCreate(true)
                                             .build("LastName", lastNameData)

// Build a chat entity object from those fields
// (to map user data to fields in a Salesforce record)
val contactEntity = ChatEntity.Builder()
    .showOnCreate(true)
    .addChatEntityField(firstNameField)
    .addChatEntityField(lastNameField)
    .build("Contact")
```

When creating the `ChatEntityField` object, you can specify whether to search for that field (`doFind`), whether the match must be exact (`isExactMatch`), and whether to create a new record if not found (`doCreate`). When creating the `ChatEntity` object, along with the name of the Salesforce object and the list of fields, you can specify whether the contact should pop up for the agent upon creation (`showOnCreate`). See the reference documentation for `ChatEntity` and `ChatEntityField`. Also refer to Chat REST API Data Types for the `Entity` and `EntityFieldsMaps` data types, which define the underlying functionality of these SDK objects.

After you've built your `ChatEntity` objects, pass them to your `ChatConfiguration` builder using the `chatEntities` method.

In Java:

```
// Create the chat configuration builder
final ChatConfiguration.Builder chatConfigurationBuilder =
  new ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                                DEPLOYMENT_ID, LIVE_AGENT_POD);

// Add user data and entities
chatConfigurationBuilder
  .chatUserData(firstNameData, lastNameData)
  .chatEntities(contactEntity);

// Build the chat configuration object
ChatConfiguration chatConfiguration = chatConfigurationBuilder.build();
```

In Kotlin:

```
// Create the chat configuration builder
val chatConfigurationBuilder = ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
    DEPLOYMENT_ID, LIVE_AGENT_POD)

// Add user data and entities
chatConfigurationBuilder
    .chatUserData(firstNameData, lastNameData)
    .chatEntities(contactEntity)

// Build the chat configuration object
val chatConfiguration = chatConfigurationBuilder.build()
```

Example: This code sample adds `FirstName`, `LastName`, `Email` to a `Contact` record and a `Subject` field to a `Case` record.

In Java:

```
// Create some hidden fields with specific values
ChatUserData firstNameData = new ChatUserData(
  "FirstName", "Jane", true);
ChatUserData lastNameData = new ChatUserData(
  "LastName", "Doe", true);
ChatUserData emailData = new ChatUserData(
  "Email", "jane.doe@salesforce.com", true);
ChatUserData subjectData = new ChatUserData(
  "Subject", "Chat Session", true);

// Map Subject to a field in a Case record
ChatEntity caseEntity = new ChatEntity.Builder()
  .showOnCreate(true)
  .linkToTranscriptField("Case")
  .addChatEntityField(
    new ChatEntityField.Builder()
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("Subject", subjectData))
  .build("Case");

// Map FirstName, LastName, and Email to fields in a Contact record
ChatEntity contactEntity = new ChatEntity.Builder()
  .showOnCreate(true)
  .linkToTranscriptField("Contact")
  .linkToAnotherSalesforceObject(caseEntity, "ContactId")
  .addChatEntityField(
    new ChatEntityField.Builder()
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("FirstName", firstNameData))
  .addChatEntityField(
    new ChatEntityField.Builder()
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("LastName", lastNameData))
  .addChatEntityField(
    new ChatEntityField.Builder()
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("Email", emailData))
  .build("Contact");

// Create the chat configuration builder
final ChatConfiguration.Builder chatConfigurationBuilder =
  new ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                                DEPLOYMENT_ID, LIVE_AGENT_POD);

// Add user data and entities
chatConfigurationBuilder
  .chatUserData(firstNameData, lastNameData, emailData, subjectData)
  .chatEntities(caseEntity, contactEntity);

// Build the chat configuration object
ChatConfiguration chatConfiguration = chatConfigurationBuilder.build();
```

In Kotlin:

```
// Create chat user data that doesn't require user interaction
val firstNameData = ChatUserData(
    "FirstName", "Jane", true)
val lastNameData = ChatUserData(
    "LastName", "Doe", true)
val emailData = ChatUserData(
    "Email", "jane.doe@salesforce.com", true)
val subjectData = ChatUserData(
    "Subject", "Chat Session", true)

// Map Subject to a field in a Case record
val caseEntity = ChatEntity.Builder()
    .showOnCreate(true)
    .linkToTranscriptField("Case")
    .addChatEntityField(
        ChatEntityField.Builder()
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("Subject", subjectData))
    .build("Case")

// Map FirstName, LastName, and Email to fields in a Contact record
val contactEntity = ChatEntity.Builder()
    .showOnCreate(true)
    .linkToTranscriptField("Contact")
    .linkToAnotherSalesforceObject(caseEntity, "ContactId")
    .addChatEntityField(
        ChatEntityField.Builder()
```

```
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("FirstName", firstNameData))
    .addChatEntityField(
        ChatEntityField.Builder()
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("LastName", lastNameData))
    .addChatEntityField(
        ChatEntityField.Builder()
          .doFind(true)
          .isExactMatch(true)
          .doCreate(true)
          .build("Email", emailData))
    .build("Contact")

// Create the chat configuration builder
val chatConfigurationBuilder = ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
    DEPLOYMENT_ID, LIVE_AGENT_POD)

// Add user data and entities
chatConfigurationBuilder
    .chatUserData(firstNameData, lastNameData, emailData, subjectData)
    .chatEntities(caseEntity, contactEntity)

// Build the chat configuration object
val chatConfiguration = chatConfigurationBuilder.build()
```

Check Agent Availability

Before starting a session, you can check the availability of your chat agents and then provide your users with more accurate expectations. For instance, when no agents are available, you can hide or disable the button to contact an agent

To check whether agents are available, create an `AgentAvailabilityClient` object and asynchronously check the `AvailabilityState` status.

In Java:

```
// Build a configuration object
ChatConfiguration chatConfiguration =
  new ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                                DEPLOYMENT_ID, LIVE_AGENT_POD)
  .build();

// Create an agent availability client
Boolean requestEstimatedWaitTime = false; // Don't request if we don't plan to use it
AgentAvailabilityClient client = ChatCore.configureAgentAvailability(chatConfiguration,
requestEstimatedWaitTime);

// Check agent availability
client.check().onResult(new Async.ResultHandler<AvailabilityState>() {
  @Override
  public void handleResult (Async<?> async, @NonNull AvailabilityState state) {

    switch (state.getStatus()) {
      case AgentsAvailable: {
        // TO DO: Handle the case where agents are available

        // Optionally, use the estimatedWaitTime to
        // show an estimated wait time until an agent
        // is available. This value is only valid
        // if you request it from the
        // configureAgentAvailability call above.
        // Estimate is returned in seconds.
        Integer ewt = state.getEstimatedWaitTime();

        break;
      }
      case NoAgentsAvailable: {
        // TO DO: Handle the case where no agents are available
        break;
      }
      case Unknown: {
        break;
      }
    }
});
```

In Kotlin:

```
// Build a configuration object
val chatConfiguration = ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
     DEPLOYMENT_ID, LIVE_AGENT_POD).build()

// Create an agent availability client
val requestEstimatedWaitTime = false // Don't request if we don't plan to use it
val client = ChatCore.configureAgentAvailability(chatConfiguration, requestEstimatedWaitTime)

// Check agent availability
client.check().onResult(object: Async.ResultHandler<AvailabilityState> {
  override fun handleResult(operation: Async<*>?, result: AvailabilityState) {
    when (result.getStatus()) {
      AvailabilityState.Status.AgentsAvailable -> {
        // TO DO: Handle the case where agents are available

        // Optionally, use the estimatedWaitTime to
        // show an estimated wait time until an agent
        // is available. This value is only valid
        // if you request it from the
        // configureAgentAvailability call above.
        // Estimate is returned in seconds.
        val ewt = result.getEstimatedWaitTime()
      }
      AvailabilityState.Status.NoAgentsAvailable -> {
        // TO DO: Handle the case where no agents are available
      }
    }
    }
})
```

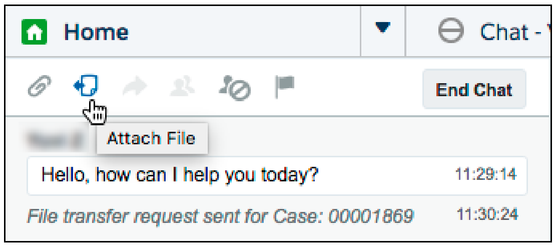
To understand the algorithm used for the estimated wait time, see the estimated wait time documentation in the Chat REST API Developer Guide.

You can also use this API to get an updated Chat server before starting a session to determine whether a server has changed for your pod. Call the `getLiveAgentPod` method from the `AvailabilityState` object you get back from the `AgentAvailabilityClient`. If the server has changed, update this configuration value in the future to prevent an unnecessary round-trip request.

Transfer File to Agent

Give users the ability to transfer files during a chat so they can share information about their issues.

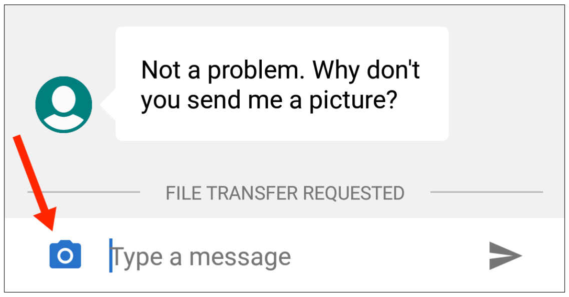
The agent can request that the user transfer a file by clicking the **Attach File** button from the Service Cloud Console.



Hello, how can I help you today? 11:29:14
File transfer request sent for Case: 00001869 11:30:24

See Transfer Files During a Chat in Salesforce Help for details about setting up this functionality in the Service Cloud Console.

With the default UI, the user sees a **FILE TRANSFER REQUESTED** message in the app and can then send a file using the camera button.


Not a problem. Why don't you send me a picture?

If you're using the default UI, no coding is necessary in your app to get this behavior.

However, if you're using the Core API on page 76, you must present your own file transfer UI based on file transfer events. Create a `FileTransferRequestListener` and pass it to the `ChatClient` using the `addFileTransferRequestListener` method. This listener gives you access to two events.

onFileTransferRequest
: The SDK calls this method when an agent requests a file transfer. You're given a `FileTransferAssistant` object, which lets you upload a file with the `uploadFile` method.

onFileTransferStatusChanged
: The SDK calls this method when the status of a file transfer has changed. You're given a `FileTransferStatus` enumerated type that describes the status of the file transfer.

You can use the following code sample as a starting point for your listener implementation.

In Java:

```
class MyFileTransferRequestListener implements FileTransferRequestListener {

  private byte[] uploadFile = new byte[]; // TO DO: File to upload
  private String fileType = "image/png";  // TO DO: File type

  @Override
  public void onFileTransferRequest (FileTransferAssistant fileTransferAssistant) {

    // TO DO: Prompt user and read the file from storage

    fileTransferAssistant.uploadFile(uploadFile, fileType);
  }

  @Override
  public void onFileTransferStatusChanged (FileTransferStatus status) {

    switch (status) {
      case Completed:
        // TO DO: File transfer completed
        break;
      case Canceled:
        // TO DO: File transfer canceled
        break;
      case Failed:
        // TO DO: File transfer failed
        break;
      case LocalError:
        // TO DO: Local error with transfer
        break;
      case Requested:
        // TO DO: File transfer requested
        // NOTE: You'll also get a call to
        // onFileTransferRequest during this state,
        // where you can handle the request and
        // then upload a file...
        break;
    }
  }
}
```

In Kotlin:

```
class MyFileTransferRequestListener : FileTransferRequestListener {

  var uploadFile = ByteArray(32768)   // TO DO: File to upload
  var fileType: String = "image/png"  // TO DO: File type

  override fun onFileTransferRequest(fileTransferAssistant: FileTransferAssistant?) {

    // TO DO: Prompt user and read the file from storage

    fileTransferAssistant?.uploadFile(uploadFile, fileType)
  }
  override fun onFileTransferStatusChanged(status: FileTransferStatus?) {

    when (status) {
      FileTransferStatus.Completed -> {
        // TO DO: File transfer completed
      }
      FileTransferStatus.Canceled -> {
        // TO DO: File transfer canceled
      }
      FileTransferStatus.Failed -> {
        // TO DO: File transfer failed
      }
      FileTransferStatus.LocalError -> {
        // TO DO: Local error with transfer
      }
      FileTransferStatus.Requested -> {
        // TO DO: File transfer requested
        // NOTE: You'll also get a call to
        // onFileTransferRequest during this state,
        // where you can handle the request and
        // then upload a file...
      }
    }
  }
}
```

Block Sensitive Data in a Chat Session

To block sending sensitive data to agents, specify a regular expression in your org's setup. When the regular expression matches text in the user's message, the matched text is replaced with customizable text before it leaves the device.

To learn more, see Block Sensitive Data in Chats.

Build Your Own UI with the Chat Core API

With the Chat Core API, you can access the functionality of Chat without a UI. This API is useful if you want to build your own UI and not use the default.

Before starting, make sure that you've already:

- Set up Service Cloud to work with Chat. See Org Setup for Chat in Lightning Experience with a Guided Flow.
- Installed the SDK. See Install the Service SDK for Android.

These steps describe how to use the Chat Core API. To use the default UI, see Quick Setup: Chat in the Service SDK.

1. To listen for agent activity, create an `AgentListener` implementation.

   From the agent listener, you can:

   - Detect when an agent has joined using `onAgentJoined(AgentInformation agentInformation)`.
   - Detect when an agent is typing a message using `onAgentIsTyping(boolean isAgentTyping)`.
   - Receive an incoming message from an agent using `onChatMessageReceived(ChatMessage chatMessage)`.

2. To listen for session state changes, create a `SessionStateListener`.

From a session state listener, you can:

- Detect when the session state changes using `onSessionStateChange(ChatSessionState state)`. For example, this method is useful for when the session disconnects (`ChatSessionState.Disconnected`).
- Detect when the session ends using `onSessionEnded(ChatEndReason endReason)`. This method is useful to know why a session ends and report the information to the user. For example, `ChatEndReason.EndedByAgent` is passed when the agent ends the session.

To learn more about using this listener, see Listen for State Changes and Events.

3. To listen for changes related to a user's position or estimated wait time in the agent queue, create a `QueueListener`.

   When a user is trying to connect to an agent, you can monitor where the user is in the queue. The estimated wait time is returned in minutes and the queue position is an integer value related to the overall agent capacity.

   When using the queue position number, the queue position is 0 if the agent capacity is greater than or equal to the number of customer requests. Otherwise, the position value represents how far the customer is from getting served by an agent.

   ```
   q = max(n - c, 0)
   ```

   Where:

   - q is the queue position
   - n is the position of the customer compared to all waiting customers
   - c is the total capacity of all agents

   For example, if the total capacity is 10, the first 10 waiting visitors have a position of 0, the 11th has a position of 1, the 12th has a position of 2, and so on.

4. To listen for Einstein bot activity, create a `ChatBotListener`.

   From a bot listener, you can:

   - Detect when you receive a persistent footer menu from the Einstein bot with `onChatFooterMenuReceived`. A footer menu is always accessible to the user and typically handles options that are not context-specific, such as "Transfer to agent."
   - Detect when you receive menu options from the Einstein bot with `onChatMenuReceived`.
   - Detect when you receive choice button options from the Einstein bot with `onChatButtonMenuReceived`.

5. Build a `ChatConfiguration` object as described in Quick Setup: Chat in the Service SDK.

   In Java:

   ```
   // Create a core configuration instance
   ChatConfiguration chatConfiguration =
     new ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                                   DEPLOYMENT_ID, LIVE_AGENT_POD)
                                   .build();
   ```

   In Kotlin:

   ```
   // Create a core configuration instance
   val chatConfiguration =
     ChatConfiguration.Builder(ORG_ID, BUTTON_ID,
                               DEPLOYMENT_ID, LIVE_AGENT_POD)
                               .build()
   ```

6. Create a `ChatCore` object with your chat configuration object.

In Java:

```
ChatCore core = ChatCore.configure(chatConfiguration);
```

In Kotlin:

```
val core = ChatCore.configure(chatConfiguration)
```

7. Create a `ChatClient`, adding your listener objects.

   Define `ChatClient` at the `Application` scope to ensure that the session is trackable throughout the application's lifetime rather than just within an `Activity`, for example.

   In Java:

```
private @Nullable ChatClient mChatClient;

// ...

core.createClient(this)
  .onResult(new Async.ResultHandler<ChatClient>() {
    @Override public void handleResult (Async<?> operation,
                                        @NonNull ChatClient chatClient) {
      mChatClient = chatClient
        .addSessionStateListener(myStateListener)
        .addAgentListener(myAgentListener)
        .addQueueListener(myQueueListener)
        .addChatBotListener(myEinsteinBotListener);
    }
});
```

   In Kotlin:

```
var mChatClient: ChatClient? = null

// ...

core.createClient(this)
  .onResult { operation, chatClient ->
    mChatClient = chatClient
      .addSessionStateListener(myStateListener)
      .addAgentListener(myAgentListener)
      .addQueueListener(myQueueListener)
      .addChatBotListener(myEinsteinBotListener)
  }
```

   When you receive a chat client instance, a session has successfully started.

8. Perform session actions with the `ChatClient` object.

   From the chat client object, you can perform the following functions:

   - Tell the agent when the user is typing a message using `setIsUserTyping(boolean isUserTyping)`.
   - Send a message to the agent using `sendChatMessage(String message)`.
   - Handle file transfer activity using `addFileTransferRequestListener(FileTransferRequestListener fileTransferRequestListener)`. See Transfer File to Agent.

- Handle Einstein bot responses with `sendFooterMenuSelection`, `sendMenuSelection`, and `sendButtonSelection`.
- End the chat session using `endChatSession()`.

Using Knowledge with the Service SDK

Add the Knowledge experience to your mobile app.

Knowledge in the Service SDK for Android

The Knowledge feature in the SDK gives you access to your org’s knowledge base directly from within your app.

Quick Setup: Knowledge in the Service SDK

To set up Knowledge in your Android app, create a configuration object that points to your knowledge base and then create a Knowledge UI client.

Knowledge as an Authenticated User

In some scenarios, you may want knowledge base access for logged-in users only. You might even have different knowledge bases for different user profiles. For these scenarios, you can use the authenticated Knowledge feature.

Add Knowledge View Additions

You can add more views, such as floating action buttons, that appear in your Knowledge activities by implementing `KnowledgeViewAddition`. Adding views is handy if you want to give your users access to other features from your support home screen. Use this technique to add action buttons to Case Management, Chat, SOS, or any other features.

Customize Knowledge Articles with JavaScript or CSS

Create a richer experience for your users by injecting custom JavaScript or CSS into your knowledge articles. For example, change the style sheet for all your articles, or add introductory content to a subset of articles.

Cache and Encryption Behavior for Knowledge

Knowledge articles are cached when they're fetched. The cached data is encrypted using AES-256 encryption.

Cache Images for Offline Access

By default, articles are cached for offline access, but the associated images are not. However, you can use the `OfflineResourceConfig` class to cache image content.

Provide Images for Categories and Articles

By implementing the `KnowledgeImageProvider`, you can provide images for use with your Knowledge categories and articles. Category images appear behind the category name when viewing a category list and they appear on the category description view. Article images appear in article lists and at the top of the article detail view.

Article Fetching with the Knowledge Core API

If you want more control over the interface, you can work with the Knowledge Core API. This API lets you query your Experience Cloud site for categories and articles and handle them as objects. You can then display the contents of these objects in your own custom UI.

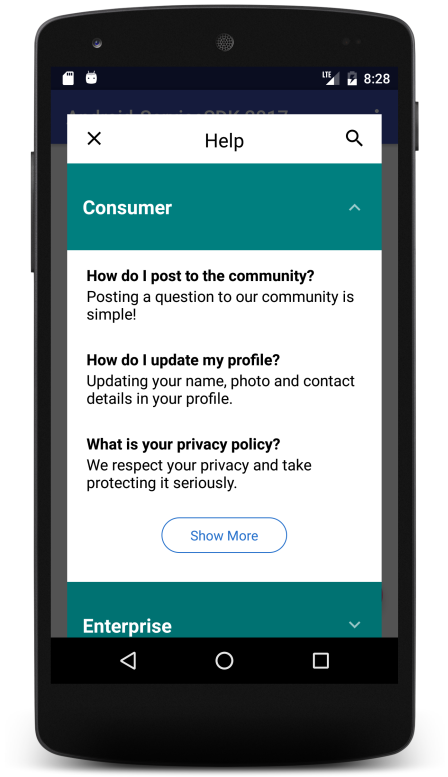
Show Knowledge Articles in a Web View

You can show an article using the `ArticleWebView` class. Use this technique to show users specific content without requiring them to browse for it.

Knowledge in the Service SDK for Android

The Knowledge feature in the SDK gives you access to your org’s knowledge base directly from within your app.

Once you point your app to your Experience Cloud site URL with the right category group and root data category, you can display your knowledge base to your users.

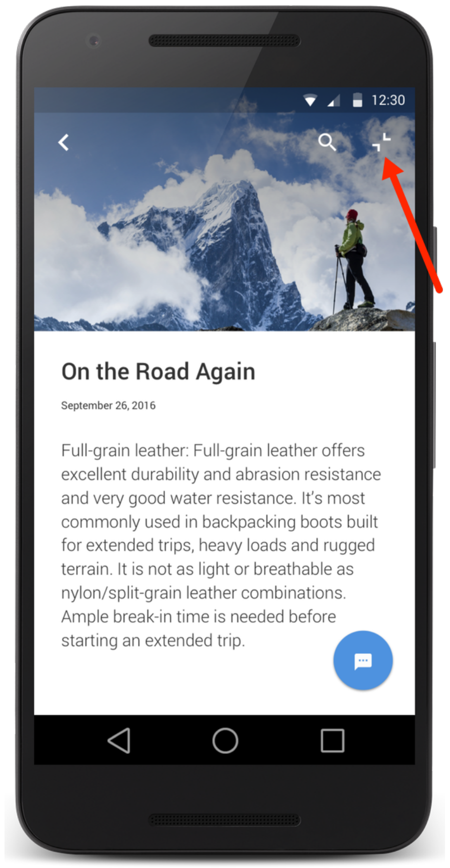


How do I post to the community?
Posting a question to our community is simple!

How do I update my profile?
Updating your name, photo and contact details in your profile.

What is your privacy policy?
We respect your privacy and take protecting it seriously.

From the knowledge home, a user can navigate through articles that are organized by category. Articles are also searchable from within the app. When a user views an article, they can minimize it using the minimize button at the top right of the article so that they can continue to navigate your app.

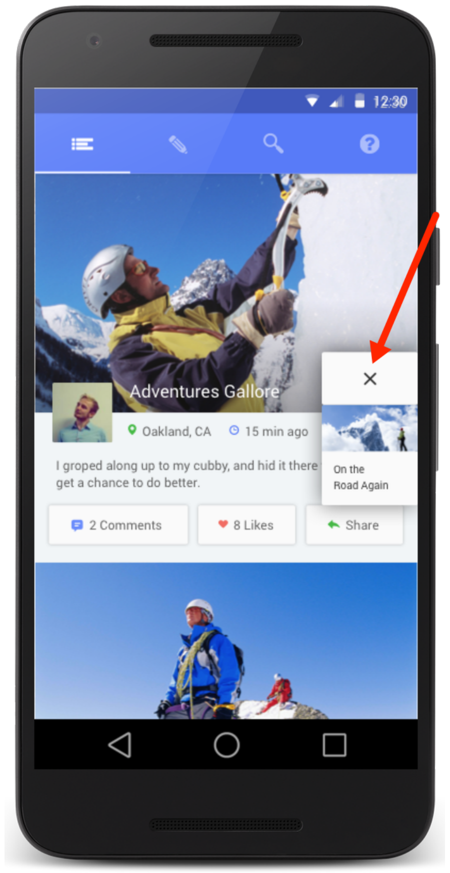


On the Road Again

September 26, 2016

Full-grain leather: Full-grain leather offers excellent durability and abrasion resistance and very good water resistance. It's most commonly used in backpacking boots built for extended trips, heavy loads and rugged terrain. It is not as light or breathable as nylon/split-grain leather combinations. Ample break-in time is needed before starting an extended trip.

Once minimized, the user can drag the article thumbnail to any part of the screen to improve visibility of the currently showing view.



Tapping the X closes the article. Tap on any other part of the thumbnail to make it full screen again.

Once you've configured Knowledge to work within your app, customize the color branding and the strings of the interface by using our UI Customization guidelines.

There are other ways to customize the Knowledge experience in your app. If you want to design your own UI, see our Knowledge Core API to access knowledge content directly. You also have finer-grained control over how images are cached and displayed, using Cache Images for Offline Access and Provide Images for Categories and Articles.

Quick Setup: Knowledge in the Service SDK

To set up Knowledge in your Android app, create a configuration object that points to your knowledge base and then create a Knowledge UI client.

Before starting, make sure that you've already:

- Set up Service Cloud to work with Knowledge. To learn more, see Cloud Setup for Knowledge.
- Installed the SDK. To learn more, see Install the Service SDK for Android.

1. Specify the site URL, the data category group, and the root data category that you want to use as the starting point for the Knowledge user experience.

   In Java:

   ```
   // Specify the Experience Cloud site url, category group, and root category
   public static final String SITE_URL = "https://your-site-url";
   ```

```
public static final String CATEGORY_GROUP = "your-category-group";
public static final String ROOT_CATEGORY = "your-root-category";
```

In Kotlin:

```
// Specify the site url, category group, and root category
val SITE_URL = "https://your-site-url"
val CATEGORY_GROUP = "your-category-group"
val ROOT_CATEGORY = "your-root-category"
```

Note: You can get the required parameters from your Salesforce org. If your Salesforce admin hasn't set up Knowledge in Service Cloud or you need more guidance, see Cloud Setup for Knowledge.

2. Create a `KnowledgeConfiguration` object using the site URL.

   In Java:

```
// Create a core configuration instance
KnowledgeConfiguration coreConfiguration =
  KnowledgeConfiguration.create(SITE_URL);
```

   In Kotlin:

```
// Create a core configuration instance
val coreConfiguration = KnowledgeConfiguration.create(SITE_URL)
```

   Note: By default, knowledge articles are cached locally but images are not cached. For guidance on setting up a configuration that caches images, see Cache Images for Offline Access.

3. Create a `KnowledgeUIConfiguration` instance using the configuration object and the root category to use as the starting place for the knowledge base.

   In Java:

```
// Create a UI configuration instance from a core instance
KnowledgeUIConfiguration uiConfiguration =
  KnowledgeUIConfiguration.create(coreConfiguration, CATEGORY_GROUP, ROOT_CATEGORY);
```

   In Kotlin:

```
// Create a UI configuration instance from core instance
val uiConfiguration =
  KnowledgeUIConfiguration.create(coreConfiguration, CATEGORY_GROUP, ROOT_CATEGORY)
```

4. (Optional) Provide images for categories and articles.

   If you'd like to provide images for Knowledge categories and Knowledge articles, implement the `KnowledgeImageProvider` interface and pass it to the `KnowledgeUIConfiguration` object.

   In Java:

```
// Specify image provider
uiConfiguration.setImageProvider(new MyImageProvider());
```

   In Kotlin:

```
// Specify image provider
uiConfiguration.setImageProvider(MyImageProvider())
```

Category images appear behind the category name when viewing a category list and they appear on the category description view. Article images appear in article lists and at the top of the article detail view.

To learn more about `KnowledgeImageProvider`, see Provide Images for Categories and Articles.

5. (Optional) Customize the interface.

   You can customize colors, strings, floating action buttons, and other aspects of the interface. You can also localize the strings into other languages. To learn more about customizations, see SDK Customizations with the Service SDK for Android.

6. Create a `KnowledgeUIClient` instance, show the UI, and maintain a reference to this instance.

   To start the Knowledge UI, call the static `configure` method, which creates a `KnowledgeUI` instance. From this instance, create a client (asynchronously) with the `createClient` method. This asynchronous call gives you a `KnowledgeUIClient` instance. You can start the UI using the `launchHome` method.

   Note: If you'd like to design your own UI and access Knowledge objects directly, see Article Fetching with the Knowledge Core API. Alternatively, if you want to use the default UI but show a specific article, call the `launchArticle` method on the `KnowledgeUIClient` instance. This method requires an `ArticleSummary` instance, which you can get from the Knowledge Core API on page 101.

   Be sure to maintain a reference to the `KnowledgeUIClient` instance for as long as you want to use the Knowledge UI. You can remove your reference to this instance using `KnowledgeUIClient.OnCloseListener` as shown in the code snippet.

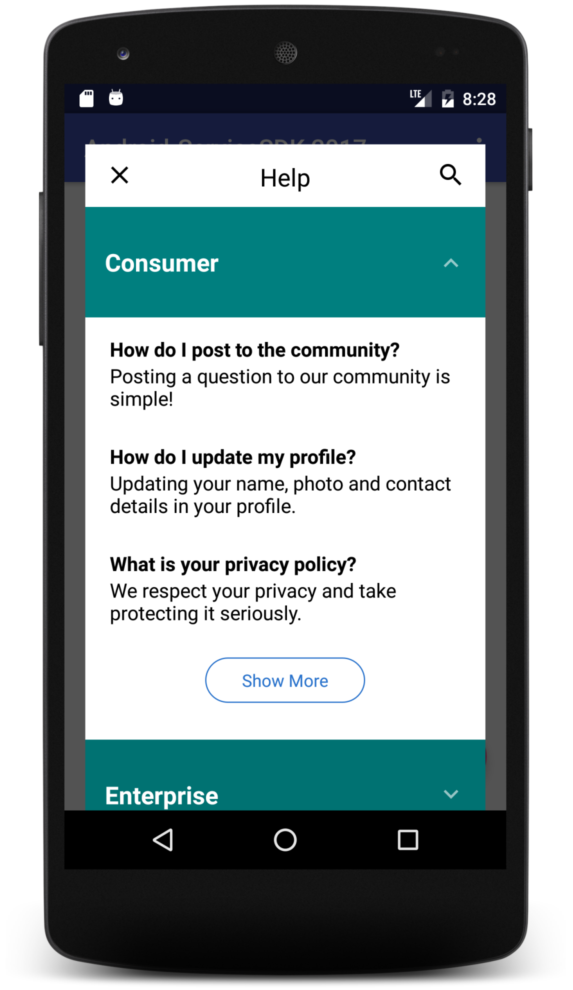
   In Java:

```
// Create a Knowledge UI instance
KnowledgeUI knowledgeUI = KnowledgeUI.configure(uiConfiguration);

// Create a client asynchronously
knowledgeUI.createClient(MainActivity.this)
  .onResult(new Async.ResultHandler<KnowledgeUIClient>() {

    @Override public void handleResult (Async<?> operation,
      KnowledgeUIClient uiClient) {

        // Maintain a reference to the UI client within your Application lifecycle
        mKnowledgeUIClient = uiClient;

        // Handle the close action
        uiClient.addOnCloseListener(new KnowledgeUIClient.OnCloseListener() {

          @Override public void onClose () {

            // Clear reference to the Knowledge UI client
            mKnowledgeUIClient = null;
          }

        });

        // Launch the UI
        uiClient.launchHome(MainActivity.this);
    }
});
```

In Kotlin:

```
// Create a Knowledge UI instance
val knowledgeUI = KnowledgeUI.configure(uiConfiguration)

// Create a client asynchronously
knowledgeUI.createClient(this@MainActivity)
    .onResult { _, uiClient ->

      // Maintain a reference to the UI client within your Application lifecycle
      mKnowledgeUIClient = uiClient

      // Handle the close action
      uiClient.addOnCloseListener {

        // Clear reference to the Knowledge UI client
        mKnowledgeUIClient = null
      }

      // Launch the UI
      uiClient.launchHome(this@MainActivity)
    }
```

Make sure that you only call the `launchHome` method one time. If you call it multiple times, multiple UIs will appear.



How do I post to the community?
Posting a question to our community is simple!

How do I update my profile?
Updating your name, photo and contact details in your profile.

What is your privacy policy?
We respect your privacy and take protecting it seriously.

Example: In this example, call `initKnowledge` to initialize the Knowledge experience, and call `startKnowledge` to launch the Knowledge experience.

In Java:

```
public class MainActivity extends Activity {

  // You should normally maintain a reference to these instances
  // within the Application lifecycle (rather than Activity lifecycle)...
  // They are placed here just for this example...
  KnowledgeUI mKnowledgeUI = null;
  KnowledgeUIClient mKnowledgeUIClient = null;

  // Specify the site url, category group, and root category
  public static final String SITE_URL = "https://your-site-url";
  public static final String CATEGORY_GROUP = "your-category-group";
  public static final String ROOT_CATEGORY = "your-root-category";

  // Call this method at init time...
  private void initKnowledge() {
    // Only initialize once
    if (mKnowledgeUI == null) {
      // Create a core configuration instance
      KnowledgeConfiguration coreConfiguration =
            KnowledgeConfiguration.create(SITE_URL);

      // Create a UI configuration instance from core instance
      KnowledgeUIConfiguration uiConfiguration =
            KnowledgeUIConfiguration.create(coreConfiguration,
                                            CATEGORY_GROUP, ROOT_CATEGORY);

      // Create a UI instance
      mKnowledgeUI = KnowledgeUI.configure(uiConfiguration);

      // Add the (optional) view addition
      KnowledgeHomeViewAddition viewAddition = new KnowledgeHomeViewAddition();
      mKnowledgeUI.viewAddition(viewAddition);
    }
  }

  // Call this method when you want to show the Knowledge UI
  private void startKnowledge() {
    // Verify that we don't already have a UI running and that we've initialized
    if (mKnowledgeUIClient == null && mKnowledgeUI != null) {

      // Create a client asynchronously
      KnowledgeUI.configure(uiConfiguration).createClient(MainActivity.this)
        .onResult(new Async.ResultHandler<KnowledgeUIClient>() {

          @Override public void handleResult (Async<?> operation,
            KnowledgeUIClient uiClient) {

              // Store reference to the Knowledge UI client
              mKnowledgeUIClient = uiClient;

              // Handle the close action
              uiClient.addOnCloseListener(new KnowledgeUIClient.OnCloseListener() {

                @Override public void onClose () {

                  // Clear reference to the Knowledge UI client
                  mKnowledgeUIClient = null;
                }
              });

              // Launch the UI
              uiClient.launchHome(MainActivity.this);
          }
      });
    }
  }

  // The rest of the Activity code goes here...
}
```

In Kotlin:

```kotlin
class MainActivity : Activity() {

  // Specify the site url, category group, and root category
  val SITE_URL = "https://your-site-url"
  val CATEGORY_GROUP = "your-category-group"
  val ROOT_CATEGORY = "your-root-category"

  var mKnowledgeUI: KnowledgeUI? = null
  var mKnowledgeUIClient:KnowledgeUIClient? = null

  private fun initKnowledge() {
    // Only initialize once
    if (mKnowledgeUI == null) {

      // Create a core configuration instance
      val coreConfiguration = KnowledgeConfiguration.create(SITE_URL)

      // Create a UI configuration instance from core instance
      val uiConfiguration = KnowledgeUIConfiguration.create(coreConfiguration,
          CATEGORY_GROUP,
          ROOT_CATEGORY)

      // Create a Knowledge UI instance
      mKnowledgeUI = KnowledgeUI.configure(uiConfiguration)
    }
  }

  private fun startKnowledge() {
    // Verify that we don't already have a UI running and that we've initialized
    if (mKnowledgeUIClient == null && mKnowledgeUI != null) {

      // Create a client asynchronously
      mKnowledgeUI!!.createClient(this@MainActivity)
          .onResult { _, uiClient ->

            // Maintain a reference to the UI client within your Application lifecycle

            mKnowledgeUIClient = uiClient

            // Handle the close action
            uiClient.addOnCloseListener {

              // Clear reference to the Knowledge UI client
              mKnowledgeUIClient = null
            }

            // Launch the UI
            uiClient.launchHome(this@MainActivity)
          }
    }
  }
  // The rest of the Activity code goes here...
}
```

Knowledge as an Authenticated User

In some scenarios, you may want knowledge base access for logged-in users only. You might even have different knowledge bases for different user profiles. For these scenarios, you can use the authenticated Knowledge feature.

Before starting, make sure that you've already:

- Set up Service Cloud to work with Knowledge. To learn more, see Cloud Setup for Knowledge.
- Installed the SDK. To learn more, see Install the Service SDK for Android.

When you activate the Knowledge interface for authenticated users, they see Knowledge content assigned to their user profile. If you do not want to authenticate users and prefer to let them see Knowledge content accessible to guest users, see Quick Setup: Knowledge in the Service SDK for instructions on accessing a public knowledge base.

Note: When using Knowledge with authenticated users, be sure that your knowledge article types are visible (set to "Read") for the desired user profile and that the knowledge articles belong to a channel that is accessible to that user. For more information, see Knowledge Article Access and Create and Edit Articles in Salesforce Help.

1. Follow authentication instructions from Authentication with the Service SDK for Android.
2. Set up Knowledge as described in Quick Setup: Knowledge in the Service SDK.

   Note: If you don't have an Experience Cloud site URL, you can use the instance URL you get back during the authentication process in place of the site URL during configuration.

   In addition to the normal setup process, be sure to build `KnowledgeConfiguration` using the `withAuthConfig` method.

   In Java:

   ```
   KnowledgeConfiguration.Builder builder =
     KnowledgeConfiguration.builder(siteUrl).withAuthConfig(myAuthProvider, myAuthUser);
   ```

   In Kotlin:

   ```
   val builder = KnowledgeConfiguration.builder(siteUrl)
                   .withAuthConfig(myAuthProvider, myAuthUser)
   ```

After you launch the UI as an authenticated user, the Knowledge home screen appears, showing articles available to users for the applicable user profile.

Add Knowledge View Additions

You can add more views, such as floating action buttons, that appear in your Knowledge activities by implementing `KnowledgeViewAddition`. Adding views is handy if you want to give your users access to other features from your support home screen. Use this technique to add action buttons to Case Management, Chat, SOS, or any other features.

Before starting, make sure that you understand how to set up and use Knowledge in your app. See Quick Setup: Knowledge in the Service SDK.

1. Implement the `KnowledgeViewAddition` interface to define the behavior of the view addition.
   a. Implement the `visibleFor` method to specify in which scenes your view appears.

      To make the view visible for a particular scene, return `true`. Otherwise, return `false`. The following scenes are available.

Scene	Description
`SCENE_HOME`	Shows the main category list. Visible when the Knowledge UI first launches.
`SCENE_ARTICLE_LIST`	Shows an expanded list of articles within a category.
`SCENE_CATEGORY_DETAIL`	Shows the category's top articles and a list of subcategories
`SCENE_ARTICLE_DETAIL`	Shows the contents of a specific article.
`SCENE_SEARCH`	Allows a user to search for articles.
`SCENE_NONE`	Transition scene used when starting or leaving the Knowledge UI.

For example, the following code shows your view on the home screen and the category detail page.

In Java:

```
@Override public boolean visibleFor (KnowledgeScene scene) {
  return (scene == SCENE_HOME || scene == SCENE_CATEGORY_DETAIL);
}
```

In Kotlin:

```
override fun visibleFor(scene: KnowledgeScene): Boolean {
  return (scene == KnowledgeScene.SCENE_HOME || scene ==
KnowledgeScene.SCENE_CATEGORY_DETAIL)
}
```

b. Implement `createView` to create the view to overlay the Knowledge UI.

The returned view is added as a new content view on the Knowledge `Activity`.

In Java:

```
@NonNull @Override public View createView (ViewGroup viewGroup, Context context) {
  return LayoutInflater.from(context).inflate(R.layout.my_addition, viewGroup, false);
}
```

In Kotlin:

```
override fun createView(viewGroup: ViewGroup, context: Context): View {
  return LayoutInflater.from(context).inflate(R.layout.my_addition, viewGroup, false)
}
```

Note: If you're using `LayoutInflator` for your `KnowledgeViewAddition` implementation, be sure to specify the third argument (`boolean attachToRoot`).

c. Implement `initView` to initialize the view returned from `createView`.

The `visible` argument specifies whether the view is visible or hidden.

In Java:

```
@Override public void initView (View view, boolean visible) {
  view.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick (View view) {
      Toast.makeText(view.getContext(),
                     "You tapped the view addition",
                     Toast.LENGTH_SHORT).show();
    }
  });

  if (!visible) {
    view.setScaleX(0);
    view.setScaleY(0);
  }
}
```

In Kotlin:

```
override fun initView(view: View, visible: Boolean) {
    view.setOnClickListener(object: View.OnClickListener {
        override fun onClick(p0: View?) {
            Toast.makeText(view.context,
                           "You tapped the view addition",
                           Toast.LENGTH_SHORT).show()
        }
    })

    if (!visible) {
        view.scaleX = 0f
        view.scaleY = 0f
    }
}
```

d. Implement `getEnterAnimator` and `getExitAnimator` to create the animators to use when entering and exiting the view.

In Java:

```
@NonNull @Override public Animator getEnterAnimator (View view) {
  AnimatorSet set = new AnimatorSet();
  set.playTogether(
            ObjectAnimator.ofFloat(view, View.SCALE_X, 1),
            ObjectAnimator.ofFloat(view, View.SCALE_Y, 1)
  );
  return set;
}

@NonNull @Override public Animator getExitAnimator (View view) {
  AnimatorSet set = new AnimatorSet();
  set.playTogether(
            ObjectAnimator.ofFloat(view, View.SCALE_X, 0),
            ObjectAnimator.ofFloat(view, View.SCALE_Y, 0)
  );
  return set;
}
```

In Kotlin:

```
override fun getEnterAnimator(view: View): Animator {
  val set = AnimatorSet()
  set.playTogether(
    ObjectAnimator.ofFloat(view, View.SCALE_X, 1f),
    ObjectAnimator.ofFloat(view, View.SCALE_Y, 1f)
  )
  return set
}

override fun getExitAnimator(view: View): Animator {
  val set = AnimatorSet()
  set.playTogether(
    ObjectAnimator.ofFloat(view, View.SCALE_X, 0f),
    ObjectAnimator.ofFloat(view, View.SCALE_Y, 0f)
  )
  return set
}
```

Note: Do not start the animator from within these methods.

2. Add your view addition to the `KnowledgeUI` instance.

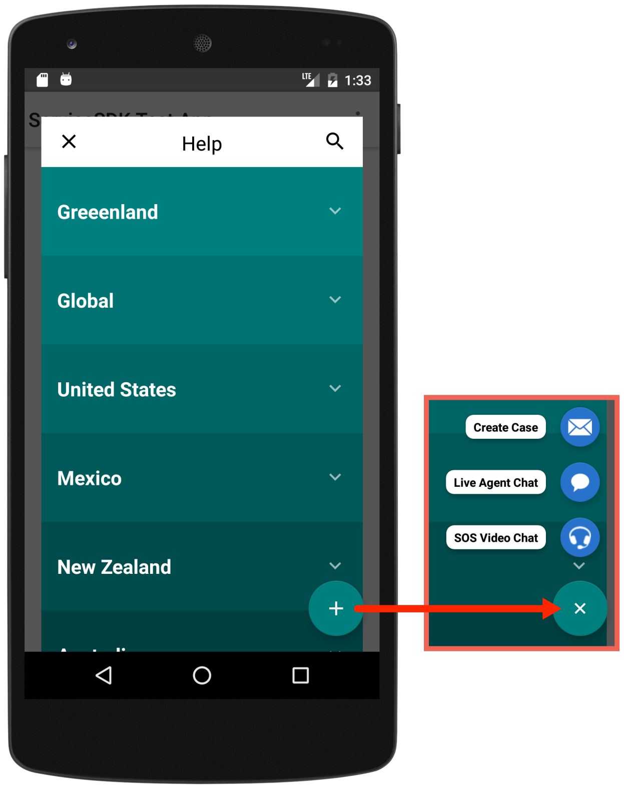
   In Java:

   ```
   mKnowledgeUI.viewAddition(new MyViewAddition());
   ```

   In Kotlin:

   ```
   mKnowledgeUI?.viewAddition(MyViewAddition())
   ```

Example: To see an example that shows you how to set up a fully featured knowledge support home experience, check out our Hello World sample app.



Customize Knowledge Articles with JavaScript or CSS

Create a richer experience for your users by injecting custom JavaScript or CSS into your knowledge articles. For example, change the style sheet for all your articles, or add introductory content to a subset of articles.

JavaScript and CSS injection can be done during the Knowledge setup process. Before starting, make sure that you understand how to set up and use Knowledge in your app. See Quick Setup: Knowledge in the Service SDK.

1. Implement the `KnowledgeJsProvider` interface if you want to inject JavaScript in your articles.

   The `getJsForArticle` method provides you with an `ArticleSummary` object. Use this article summary to determine whether to inject JavaScript into an article.

2. Implement the `KnowledgeCssProvider` interface if you want to inject CSS in your articles.

   The `getCssForArticle` method provides you with an `ArticleSummary` object. Use this article summary to determine whether to inject CSS into an article. Refer to the `KnowledgeCssProvider` Javadoc for tips on which selectors are relevant to knowledge articles.

3. When creating your `KnowledgeUIConfiguration` instance, call `setJsProvider` or `setCssProvider` with an instance to your provider implementation.

In Java:

```
// Create a UI configuration instance from a core instance
KnowledgeUIConfiguration uiConfiguration =
  KnowledgeUIConfiguration.create(coreConfiguration, CATEGORY_GROUP, ROOT_CATEGORY);
```

```
// Add your JS provider
uiConfiguration.setJsProvider(myJsProvider);

// Add your CSS provider
uiConfiguration.setCssProvider(myCssProvider);
```

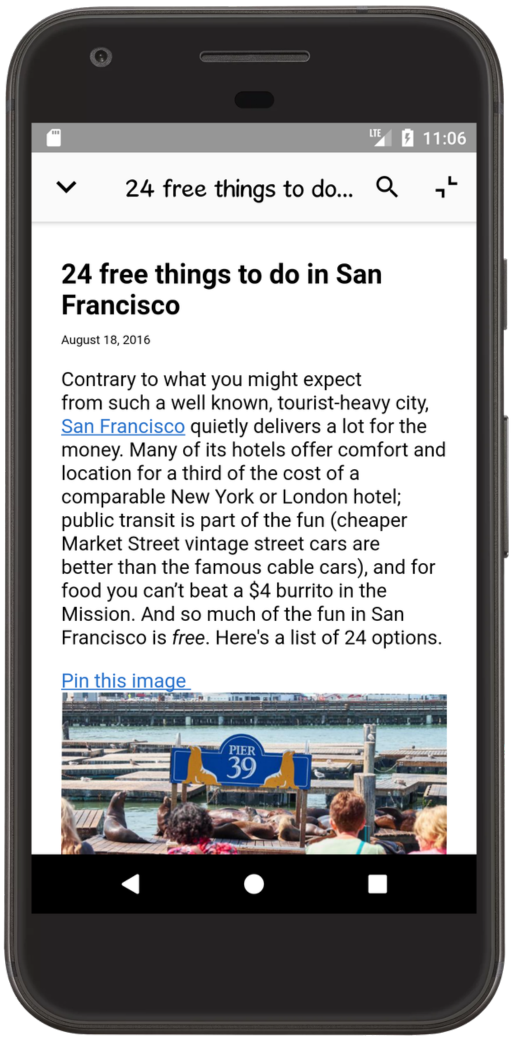
In Kotlin:

```
// Create a UI configuration instance from core instance
val uiConfiguration =
  KnowledgeUIConfiguration.create(coreConfiguration, CATEGORY_GROUP, ROOT_CATEGORY)
```

```
// Add your JS provider
uiConfiguration.setJsProvider(myJsProvider)

// Add your CSS provider
uiConfiguration.setCssProvider(myCssProvider)
```

Example: This example injects custom CSS to all articles and injects custom JavaScript only to articles whose title contains "San Francisco." You can apply these concepts to your own use cases.

Let's run this example on the following articles.

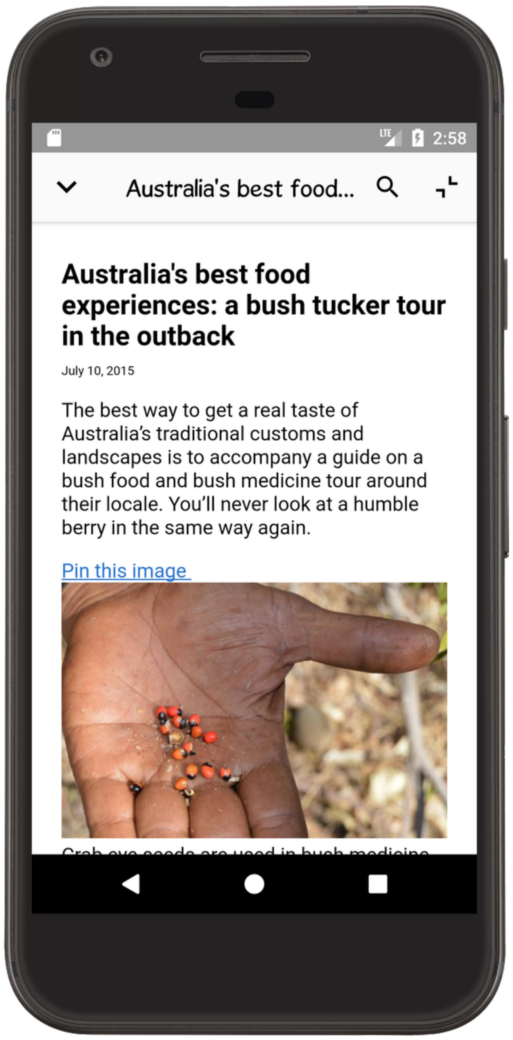


24 free things to do in San Francisco

August 18, 2016

Contrary to what you might expect from such a well known, tourist-heavy city, San Francisco quietly delivers a lot for the money. Many of its hotels offer comfort and location for a third of the cost of a comparable New York or London hotel; public transit is part of the fun (cheaper Market Street vintage street cars are better than the famous cable cars), and for food you can't beat a $4 burrito in the Mission. And so much of the fun in San Francisco is *free*. Here's a list of 24 options.

Pin this image



Australia's best food experiences: a bush tucker tour in the outback

July 10, 2015

The best way to get a real taste of Australia's traditional customs and landscapes is to accompany a guide on a bush food and bush medicine tour around their locale. You'll never look at a humble berry in the same way again.

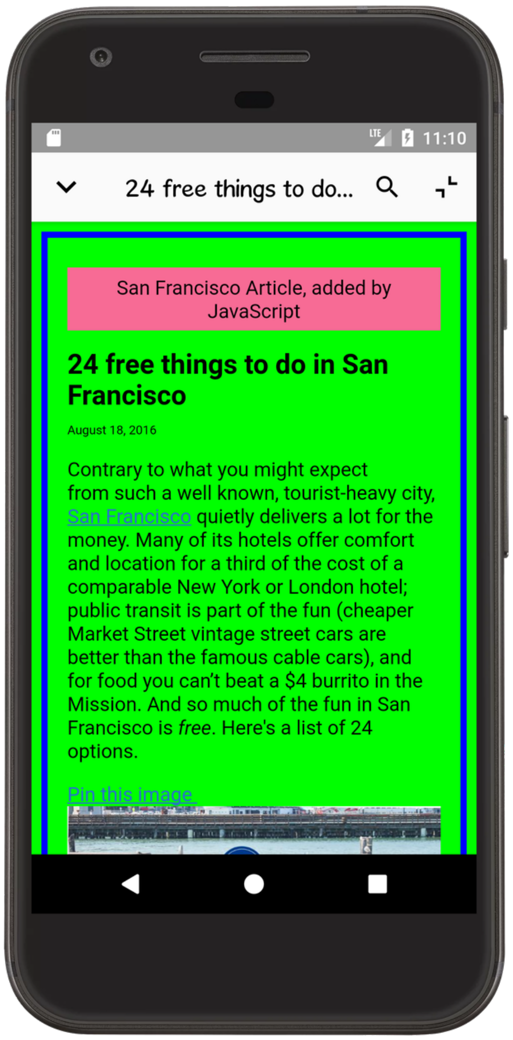
Pin this image

Crab eye seeds are used in bush medicine

We want to make 2 changes to the articles.

- A global CSS change to give all articles a vivid green background and thick blue border.
- A JavaScript change to insert a bright pink paragraph at the top of all articles whose title includes "San Francisco."

After the example runs, you see the following changes.



11:10

24 free things to do...

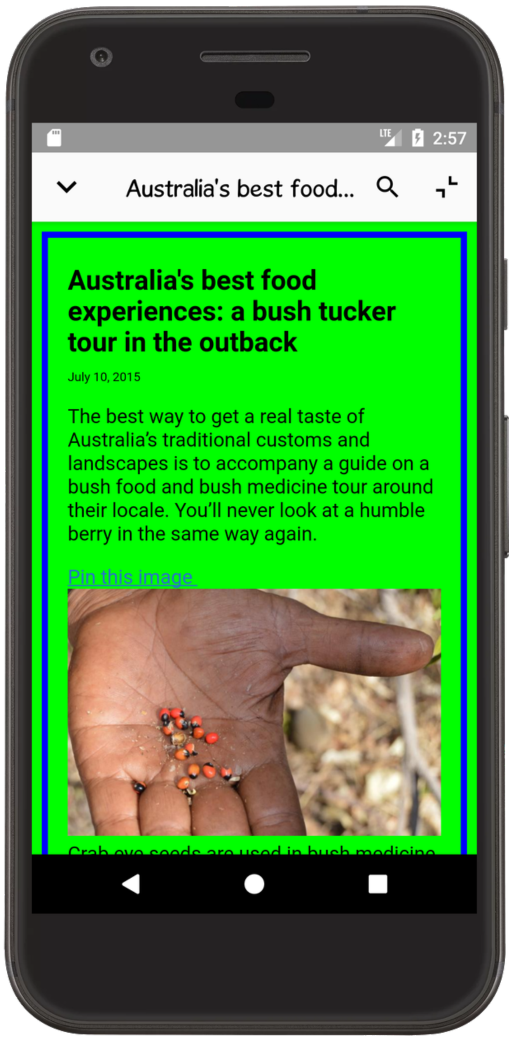
San Francisco Article, added by JavaScript

24 free things to do in San Francisco

August 18, 2016

Contrary to what you might expect from such a well known, tourist-heavy city, San Francisco quietly delivers a lot for the money. Many of its hotels offer comfort and location for a third of the cost of a comparable New York or London hotel; public transit is part of the fun (cheaper Market Street vintage street cars are better than the famous cable cars), and for food you can't beat a $4 burrito in the Mission. And so much of the fun in San Francisco is *free*. Here's a list of 24 options.

Pin this image



2:57

Australia's best food...

Australia's best food experiences: a bush tucker tour in the outback

July 10, 2015

The best way to get a real taste of Australia's traditional customs and landscapes is to accompany a guide on a bush food and bush medicine tour around their locale. You'll never look at a humble berry in the same way again.

Pin this image

Crab eye seeds are used in bush medicine

To implement the green background color and thick blue border, we use the following CSS in a file named `custom_global_style.css`.

```
body {
   background: #00ff00;
   border: thick solid #0000ff;
}
```

Let's create a `KnowledgeCssProvider` implementation to inject our custom CSS into every article.

```
public class CssProvider implements KnowledgeCssProvider {

  private final Context mContext;

  public CssProvider (Context context) {
    mContext = context;
  }

  @Override public String getCssForArticle (ArticleSummary articleSummary) {

    // Grab our custom CSS that applies to all articles...
    String globalCss = AssetUtil.readAssetFile(mContext, "custom_global_style.css");

    return globalCss;
  }
}
```

To add the introductory paragraph to articles, we use the following JavaScript in a file named `custom_sanfran_javascript.js`.

```
var testNode = document.createElement("p");
testNode.innerText = "San Francisco Article, added by JavaScript";
testNode.style.color = "#000000"
testNode.style.background = "#f76b95"
testNode.style.padding = "5pt"
testNode.style.textAlign = "center"
document.body.insertBefore(testNode, document.body.firstChild);
```

This time, let's inspect what we receive in the article summary so that our `KnowledgeJsProvider` implementation only injects JavaScript to articles whose title contains "San Francisco." (You can use the same technique to inspect the article summary in your `KnowledgeCssProvider` implementation.)

```
public class JsProvider implements KnowledgeJsProvider {

  private final Context mContext;

  public JsProvider (Context context) {
    mContext = context;
  }

  @Override public String getJsForArticle (ArticleSummary articleSummary) {
    String articleJs = "";

    // Does the article title contain "San Francisco"?
    if (articleSummary.getTitle().contains("San Francisco")) {

      // If so, then grab our custom JavaScript...
      articleJs = AssetUtil.readAssetFile(mContext, "custom_sanfran_javascript.js");
    }
    return articleJs;
  }
}
```

Now we're ready to add our providers when we set up Knowledge.

```
// Create a core configuration instance
KnowledgeConfiguration coreConfiguration =
  KnowledgeConfiguration.create(SITE_URL);
```

```
// Create a UI configuration instance from a core instance
KnowledgeUIConfiguration uiConfiguration =
  KnowledgeUIConfiguration.create(coreConfiguration, CATEGORY_GROUP, ROOT_CATEGORY);
```

```
// Add your JS provider
uiConfiguration.setJsProvider(new JsProvider(this));
```

```
// Add your CSS provider
uiConfiguration.setCssProvider(new CssProvider(this));
```

And that's it! This app now injects custom CSS and JavaScript into knowledge articles.

Cache and Encryption Behavior for Knowledge

Knowledge articles are cached when they're fetched. The cached data is encrypted using AES-256 encryption.

To delete locally cached content, call `deleteCache` on the `KnowledgeCore` object.

In Java:

```
KnowledgeCore.deleteCache(myContext);
```

In Kotlin:

```
KnowledgeCore.deleteCache(myContext)
```

Note: Article images are not cached by default. To cache images, see Cache Images for Offline Access. Cached images are encrypted.

Cache Images for Offline Access

By default, articles are cached for offline access, but the associated images are not. However, you can use the `OfflineResourceConfig` class to cache image content.

You can store and retrieve cached images by creating an `OfflineResourceConfig` instance and passing it to the `KnowledgeConfiguration` instance during the setup process. See Quick Setup: Knowledge in the Service SDK for setup instructions using the default UI and Article Fetching with the Knowledge Core API for direct access to knowledge objects.

The `OfflineResourceConfig` class allows you to specify the maximum cache size and the number of concurrent requests using the builder design pattern.

In Java:

```
OfflineResourceConfig offlineConfig = OfflineResourceConfig.Builder()
  .maxSize(60 * 1024 * 1024)    // 60 MB cache
  .concurrentRequests(8)        // 8 concurrent requests
  .build();
```

In Kotlin:

```
var offlineConfig = OfflineResourceConfig.Builder()
    .maxSize(60 * 1024 * 1024)    // 60 MB cache
    .concurrentRequests(8)        // 8 concurrent requests
    .build()
```

Once you've instantiated an `OfflineResourceConfig`, you can pass that instance to `KnowledgeConfiguration` using the `offlineResourceConfig` method during construction.

In Java:

```
KnowledgeConfiguration.builder(SITE_URL).offlineResourceConfig(offlineConfig).build();
```

In Kotlin:

```
KnowledgeConfiguration.builder(SITE_URL).offlineResourceConfig(offlineConfig).build()
```

From here, you can proceed with the instructions for setting up the default Knowledge UI (Quick Setup: Knowledge in the Service SDK) or for direct access to the Knowledge objects (Article Fetching with the Knowledge Core API).

When using `OfflineResourceConfig`, images are automatically cached locally. When the cache reaches its max size, oldest content is removed.

To delete existing cached images, use `KnowledgeClient.getResourceCacher().getCache().clear()`.

Provide Images for Categories and Articles

By implementing the `KnowledgeImageProvider`, you can provide images for use with your Knowledge categories and articles. Category images appear behind the category name when viewing a category list and they appear on the category description view. Article images appear in article lists and at the top of the article detail view.

The `KnowledgeImageProvider` interface has one method for article images and one method for category images:

```
public interface KnowledgeImageProvider {
  Drawable getImageForArticle (Context context, ArticleSummary article);
  Drawable getImageForDataCategory (Context context, DataCategorySummary dataCategory);
}
```

The SDK calls these methods whenever it attempts to load an image for a given data category or article. `getImageForArticle` passes you an `ArticleSummary` object and `getImageForDataCategory` passes you a `DataCategorySummary` object.

You'll need a way to correlate articles and categories with images. Data categories can be uniquely identified with a `DataCategorySummary` object by using `getName` to get the category name (for example, "Asia"). Articles can be uniquely identified with an `ArticleSummary` object by using `getArticleId` to get the article ID (for example, "000005256").

Once you've implemented the `KnowledgeImageProvider` interface, pass an instance of your image provider class to the Knowledge UI configuration object.

In Java:

```
KnowledgeUIConfiguration uiConfiguration =
  KnowledgeUIConfiguration.create(coreConfiguration, categoryGroup, rootCategory)
    .setImageProvider(new MyImageProvider());
```

In Kotlin:

```
val uiConfiguration = KnowledgeUIConfiguration
  .create(coreConfiguration, categoryGroup, rootCategory)
    .setImageProvider(MyImageProvider())
```

Refer to Quick Setup: Knowledge in the Service SDK for more information about starting the Knowledge interface.

Example: Since the `KnowledgeImageProvider` interface does not specify how the images are sourced, this example illustrates how to load them from disk as project assets. For performance reasons, this example uses Android's `LruCache` API for efficient loading of assets for future requests. This method helps avoid high UI latency and excess memory consumption.

```
/**
 * KnowledgeImageProvider implementation that loads images used by Knowledge UI.
 * This implementation assumes that images are stored as PNG files in the assets
 * directory of the project. Data Category images are stored in the "categories"
 * subfolder with the category name as the file name. Article images are stored
 * in the "articles" subfolder with the article number as the file name.
 */
public class ImageProvider implements KnowledgeImageProvider {

  private final LruCache<String, BitmapDrawable> mCategoryImageCache;
  private final LruCache<String, BitmapDrawable> mArticleImageCache;

  ImageProvider () {
    // The maximum amount of memory available to the application in kilobytes.
    final int maxMemory = ((int) Runtime.getRuntime().maxMemory() / 1024);

    // Use 1/8th the maximum available memory to cache category images. Article
    // images can use less because we don't display as many of them at once.
    final int categoryCacheSize = maxMemory / 8;
    final int articleCacheSize = maxMemory / 16;

    // Create an LruCache that is bounded by the maximum available amount of memory.
    mCategoryImageCache = makeCache(categoryCacheSize);
    mArticleImageCache = makeCache(articleCacheSize);
  }

  //-----------------------------------------------------------------------------
  // KnowledgeImageProvider API
  //-----------------------------------------------------------------------------

  @Override public Drawable getImageForArticle (Context context,
                                              ArticleSummary article) {
    String name = "articles/" + article.getArticleNumber() + ".png";
    return getDrawable(context, name, mArticleImageCache);
  }

  @Override
  public Drawable getImageForDataCategory (Context context,
                                          DataCategorySummary dataCategory) {
    String name = "categories/" + dataCategory.getName().toLowerCase() + ".png";
    return getDrawable(context, name, mCategoryImageCache);
  }

  //--------------------------------------------------------------------------------
  // Helpers
  //--------------------------------------------------------------------------------

  private static LruCache<String, BitmapDrawable> makeCache (int size) {
    return new LruCache<String, BitmapDrawable>(size) {
      @Override protected int sizeOf (String key, BitmapDrawable drawable) {
        return drawable.getBitmap().getByteCount() / 1024;
      }
    };
  }

  private static BitmapDrawable getDrawable (Context context, String name,
                                            LruCache<String, BitmapDrawable> cache)
{
    // Try to fetch from the in-memory cache.
    BitmapDrawable drawable = cache.get(name);

    // If it's not there then load it from disk.
    if (drawable == null) {
      drawable = loadBitmapFromDisk(context, name);
      if (drawable != null) {
        cache.put(name, drawable);
      }
    }

    return drawable;
  }

  private static BitmapDrawable loadBitmapFromDisk (Context context, String name) {
    AssetManager assets = context.getAssets();

    InputStream stream = null;
    try {
      stream = assets.open(name);

      // Lower the sample size so the images take less memory.
      BitmapFactory.Options options = new BitmapFactory.Options();
      options.inSampleSize = 2;

      Bitmap bitmap = BitmapFactory.decodeStream(stream, null, options);
      return new BitmapDrawable(context.getResources(), bitmap);
    } catch (IOException e) {
      return null;
    } finally {
      if (stream != null) {
        try {
          stream.close();
        } catch (IOException e) {
          // Log the stream closing error.
        }
      }
    }
  }
}
```

Article Fetching with the Knowledge Core API

If you want more control over the interface, you can work with the Knowledge Core API. This API lets you query your Experience Cloud site for categories and articles and handle them as objects. You can then display the contents of these objects in your own custom UI.

To access Knowledge Core objects, instantiate a `KnowledgeCore` client by calling the `createClient` method.

In Java:

```
// Create a core configuration instance
String SITE_URL = "https://ExperienceCloudSitesSubdomainName.force.com";
KnowledgeConfiguration config = KnowledgeConfiguration.create(SITE_URL);

// Create a Knowledge Core object
```

```
KnowledgeCore.configure(config).createClient(getContext())
  .onResult(new Async.ResultHandler<KnowledgeClient>() {

    @Override public void handleResult (Async<?> operation, @NonNull KnowledgeClient client)
 {
      // TO DO: Use the client instance
    }
  }
);
```

Note: If you're using enhanced domains, your org's Experience Cloud sites URL is different. For details, see My Domain URL Formats in Salesforce Help.

In Kotlin:

```
// Create a core configuration instance
val SITE_URL = "https://ExperienceCloudSitesSubdomainName.force.com"
val config = KnowledgeConfiguration.create(SITE_URL)

// Create a Knowledge Core object
KnowledgeCore.configure(config)
    .createClient(this.applicationContext)
    .onResult(object: Async.ResultHandler<KnowledgeClient> {

      override fun handleResult(operation: Async<*>?, result: KnowledgeClient) {
        // TO DO: Use the client instance
      }
    })
```

Note: You can get the site URL for the `create` method from your Salesforce org. If your Salesforce admin hasn't set up Knowledge in Service Cloud or you need more guidance, see Cloud Setup for Knowledge.

Once you've launched the Knowledge Core client, you can query for categories and articles.

Submitting Queries

You can query for data by calling the overloaded `submit` method on `KnowledgeClient` with a request object that contains the criteria for your specific request. The submit method asynchronously returns an object representing the data your requested.

Table 4: Submitting a Request

Request Object	Response Type	Description
DataCategoryGroupListRequest	DataCategoryGroupList	Represents a list of data category groups, which are groups that contain a hierarchy of data categories within them.
DataCategoriesRequest	DataCategoryList	A hierarchical list of data categories.
ArticleListRequest	ArticleList	A list of articles with their summaries.
ArticleDetailRequest	ArticleDetails	Complete contents of an article.

You create a request by using the builder method on a particular request object.

To request a list of the high-level **data category groups** within your site:

```
// Request a list of category groups in Java
DataCategoryGroupListRequest request = DataCategoryGroupListRequest.builder().build();

// Request a list of category groups in Kotlin
var request = DataCategoryGroupListRequest.builder().build()
```

To request a single data category group by name ("Travel" in this case):

```
// Request a data category group in Java
DataCategoryGroupRequest request = DataCategoryGroupRequest.builder("Travel").build();

// Request a data category group in Kotlin
var request = DataCategoryGroupRequest.builder("Travel").build()
```

To request a list of **data categories** within a data category group or within a parent data category, build a `DataCategoriesRequest` object with the name of the data category group *and* the name of a data category:

```
// Request data categories from a group in Java
String categoryGroup = "Travel";
String rootCategory = "Canada";
DataCategoriesRequest request =
  DataCategoriesRequest.builder(categoryGroup, rootCategory).build();

// Request data categories from a group in Kotlin
val categoryGroup = "Travel"
val rootCategory = "Canada"
var request =
  DataCategoriesRequest.builder(categoryGroup, rootCategory).build()
```

You can also use the term "All" to grab all subcategories:

```
// Request all subcategories in Java
DataCategoriesRequest request = DataCategoriesRequest.builder(categoryGroup, "All").build();

// Request all subcategories in Kotlin
var request = DataCategoriesRequest.builder(categoryGroup, "All").build()
```

When querying for **articles**, you have a few ways to optionally refine your query using `ArticleListRequest`:

```
// Query for articles in Java
DataCategorySummary
  canadaDataCategorySummary = // Obtained from DataCategoryList.getDataCategories()
ArticleListRequest request = ArticleListRequest.builder()
     .dataCategory(categoryGroup, canadaDataCategorySummary.getName())
     .pageNumber(1)
     .pageSize(10)
     .searchTerm("Hockey")
     .build();

// Query for articles in Kotlin
var canadaDataCategorySummary: DataCategorySummary =
     // Obtained from DataCategoryList.dataCategories
var request = ArticleListRequest.builder()
     .dataCategory(categoryGroup, canadaDataCategorySummary.name)
     .pageNumber(1)
```

```
    .pageSize(10)
    .searchTerm("Hockey")
    .build()
```

Specifically, you can use the following query parameters:

pageNumber
The page number of the results. This value is 1-based. Default value is 1.

pageSize
The number of articles per page. Default value is 3.

searchTerm
The search term used to refine the search. By default, no search term is used.

sortBy
What field you want to use for sorting. Possible values are: `SORT_BY_LAST_PUBLISHED_DATE`, `SORT_BY_TITLE`, and `SORT_BY_VIEW_SCORE`. Default value is `SORT_BY_LAST_PUBLISHED_DATE`.

sortOrder
Whether you want to sort results in descending (`SORT_DESC`) or ascending (`SORT_ASC`) order. Default is `SORT_DESC`.

The results are sorted by last modified date, in descending order.

Once you have an `ArticleList`, you can use the `ArticleSummary` from this list to request the **contents of an article**.

```
// Request article content in Java
ArticleSummary
  hockeyInCanada = // Obtained from ArticleList.getArticles()
ArticleDetailRequest request = ArticleDetailRequest.builder(hockeyInCanada).build();

// Request article content in Kotlin
var hockeyInCanada: ArticleSummary = // Obtained from ArticleList.getArticles()
var request = ArticleDetailRequest.builder(hockeyInCanada).build()
```

Alternatively, you can fetch an article directly using the article ID. This method is useful if you want to retrieve a specific article and there's no need to query for an article list.

```
// Request article by ID in Java
String articleID = "18_DIGIT_ID_GOES_HERE";
ArticleDetailRequest request = ArticleDetailRequest.builder(articleID).build();

// Request article by ID in Kotlin
val articleID = "18_DIGIT_ID_GOES_HERE"
var request = ArticleDetailRequest.builder(articleID).build()
```

To submit a request, call the overloaded `submit` method on `KnowledgeClient` and asynchronously handle the query response. The following example illustrates how to submit a request once you've built a request object:

```
// Submit a request in Java
knowledgeClient.submit(request).onResult(new Async.ResultHandler<RESPONSE_TYPE_GOES_HERE>()
 {

  @Override
  public void handleResult (Async<?> operation, @NonNull RESPONSE_TYPE_GOES_HERE result)
{
    // TO DO: Inspect the response object
  }
}).onComplete(new Async.CompletionHandler() {
  // TO DO: Handle completion
});

// Submit a request in Kotlin
knowledgeClient.submit(request).onResult(object: Async.ResultHandler<RESPONSE_TYPE_GOES_HERE>
 {

  override fun handleResult(operation: Async<*>?, result: RESPONSE_TYPE_GOES_HERE) {
    // TO DO: Inspect the response object
  }
})
```

Caching Query Results

Every request object has a `cacheResults` builder method, which you can use to **cache the results** of the query. By default, this value is set to `true`. Although the article text is cached, images are not automatically cached. To **cache the images** for offline access, you'll need to use `OfflineResourceConfig`. To learn more, see Cache Images for Offline Access.

Handling Queries

Handle the result of a request using the `Async` interface and its associated handler interfaces: `ResultHandler`, `CompletionHandler`, and `ErrorHandler`. Alternatively you can implement the broad `Handler` interface which itself inherits from all three interfaces. For convenience, the `Async<T>` methods that accept handlers as parameters return the `Async<T>` instance so you can chain the syntax.

Keep in mind that a `ResultHandler` instance can be called multiple times before the `Async<T>` operation is complete, and some of the results can be duplicates. All Knowledge objects have unique IDs so you can check for duplicates.

In Java:

```
public interface Async<T> {
  Async<T> onResult(ResultHandler<? super T> handler);
  Async<T> onComplete(CompletionHandler handler);
  Async<T> onError(ErrorHandler handler);
  Async<T> removeResultHandler (ResultHandler<? super T> handler);
  Async<T> removeCompletionHandler (CompletionHandler handler);
  Async<T> removeErrorHandler (ErrorHandler handler);
  void cancel();
  boolean inProgress();
  boolean isComplete();
  boolean isCancelled();
  boolean hasFailed();

  interface ResultHandler<T> {
    void handleResult (Async<?> operation, @NonNull T result);
  }

  interface CompletionHandler {
    void handleComplete (Async<?> operation);
  }
  interface ErrorHandler {
    void handleError (Async<?> operation, @NonNull Throwable throwable);
  }

  interface Handler<T> extends ResultHandler<T>, CompletionHandler, ErrorHandler {

  }
}
```

In Kotlin:

```
interface Async<T> {
  fun onResult(handler: ResultHandler<in T>): Async<T>
  fun onComplete(handler: CompletionHandler): Async<T>
  fun onError(handler: ErrorHandler): Async<T>
  fun removeResultHandler(handler: ResultHandler<in T>): Async<T>
  fun removeCompletionHandler(handler: CompletionHandler): Async<T>
  fun removeErrorHandler(handler: ErrorHandler): Async<T>
  fun cancel()
  fun inProgress(): Boolean
  val isComplete: Boolean
  val isCancelled: Boolean
  fun hasFailed(): Boolean

  interface ResultHandler<T> {
    fun handleResult(operation: Async<*>, result: T)
  }

  interface CompletionHandler {
    fun handleComplete(operation: Async<*>)
  }

  interface ErrorHandler {
    fun handleError(operation: Async<*>, throwable: Throwable)
  }

  interface Handler<T> : ResultHandler<T>, CompletionHandler, ErrorHandler {
  }
}
```

Displaying Articles

To display Knowledge articles in a web view, see Show Knowledge Articles in a Web View.

To display an article in the default UI, call the `launchArticle` method on the `KnowledgeUIClient` instance. This method requires the `ArticleSummary` object that you obtained from an `ArticleListRequest`. To learn more about showing the default UI, see Quick Setup: Knowledge in the Service SDK.

Examples

Fetching **data category groups**:

```
DataCategoryGroupListRequest dataCategoryGroupRequest =
  DataCategoryGroupListRequest.builder().build();
```

```
knowledgeClient.
  submit(dataCategoryGroupRequest).onResult(new Async.ResultHandler<DataCategoryGroupList>()
 {

  @Override
  public void handleResult (Async<?> operation, @NonNull DataCategoryGroupList result) {
    // The result is a DataCategoryGroupList objects, which contains DataCategoryGroup
objects
  }
}).onComplete(new Async.CompletionHandler() {

  @Override public void handleComplete (Async<?> operation) {
    // Called when the operation is complete
  }
}).onError(new Async.ErrorHandler() {

  @Override public void handleError (Async<?> operation, @NonNull Throwable throwable) {
    // Called if an error has been encountered
  }
});
```

Fetching **data categories** within the "Canada" category of "Travel":

```
String categoryGroup = "Travel";
String rootCategory = "Canada";
DataCategoriesRequest dataCategoriesRequest =
  DataCategoriesRequest.builder(categoryGroup, rootCategory).build();
knowledgeClient.
  submit(dataCategoriesRequest).onResult(new Async.ResultHandler<DataCategoryList>() {

  @Override public void handleResult (Async<?> operation, @NonNull DataCategoryList result)
 {
    // The result is a DataCategoryList object, which contains a list of DataCategorySummary
 objects
  }
}).onComplete(new Async.CompletionHandler() {

  @Override public void handleComplete (Async<?> operation) {
    // Called when the operation is complete
  }
}).onError(new Async.ErrorHandler() {

  @Override public void handleError (Async<?> operation, @NonNull Throwable throwable) {
    // Called if an error has been encountered
  }
});
```

Fetching **articles** within a data category:

```
String categoryGroup = "Travel";
String categoryName = "Canada";
ArticleListRequest articleListRequest = ArticleListRequest.builder()
    .dataCategory(categoryGroup, categoryName)
    .pageNumber(1)
    .pageSize(10)
    .queryMethod(ArticleListRequest.QUERY_BELOW)
    .searchTerm("Hockey")
    .build();
knowledgeClient.submit(articleListRequest).onResult(new Async.ResultHandler<ArticleList>()
 {

  @Override public void handleResult (Async<?> operation, @NonNull ArticleList result) {
    // The result is an ArticleList object which contains a list of ArticleSummary objects

  }
}).onComplete(new Async.CompletionHandler() {

  @Override public void handleComplete (Async<?> operation) {
    // Called when the operation is complete
  }
}).onError(new Async.ErrorHandler() {

  @Override public void handleError (Async<?> operation, @NonNull Throwable throwable) {
    // Called if an error has been encountered
  }
});
```

Fetching the **contents of an article**:

```
ArticleSummary articleSummary = // Obtained from ArticleList.getArticles()
ArticleDetailRequest articleDetailRequest =
ArticleDetailRequest.builder(articleSummary).build();
knowledgeClient.submit(articleDetailRequest).onResult(new
Async.ResultHandler<ArticleDetails>() {

  @Override public void handleResult (Async<?> operation, @NonNull ArticleDetails result)
 {
    // The result is an ArticleDetails object that contains the contents of the article
  }
}).onComplete(new Async.CompletionHandler() {

  @Override public void handleComplete (Async<?> operation) {
    // Called when the operation is complete
  }
}).onError(new Async.ErrorHandler() {

  @Override public void handleError (Async<?> operation, @NonNull Throwable throwable) {
    // Called if an error has been encountered
  }
});
```

Fetching an **article using the article ID**:

```
String articleID = "kA0P000000004SzKAI";
ArticleDetailRequest request = ArticleDetailRequest.builder(articleID).build();

knowledgeClient.submit(request).onResult(new Async.ResultHandler<ArticleDetails>() {

  @Override public void handleResult (Async<?> operation, @NonNull ArticleDetails result)
 {
    // The result is an ArticleDetails object that contains the contents of the article
  }
}).onComplete(new Async.CompletionHandler() {

  @Override public void handleComplete (Async<?> operation) {
    // Called when the operation is complete
  }
}).onError(new Async.ErrorHandler() {

  @Override public void handleError (Async<?> operation, @NonNull Throwable throwable) {
    // Called if an error has been encountered
  }
});";
```

Show Knowledge Articles in a Web View

You can show an article using the `ArticleWebView` class. Use this technique to show users specific content without requiring them to browse for it.

These instructions allow you to put a web view containing an article into your existing Activity. If you want to use the default user interface provided by the SDK, see Quick Setup: Knowledge in the Service SDK. If you want to build the entire user interface from scratch, see Article Fetching with the Knowledge Core API.

1. Create an `ArticleWebView` in your layout.

   ```
   <com.salesforce.android.knowledge.ui.ArticleWebView
           android:id="@+id/article_webview"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content" />
   ```

2. To show a Knowledge article, instantiate a `KnowledgeCore` client by calling the `createClient` method.

   In Java:

   ```
   // Create a core configuration instance
   String SITE_URL = "https://ExperienceCloudSitesSubdomainName.force.com";
   KnowledgeConfiguration config = KnowledgeConfiguration.create(SITE_URL);

   // Create a Knowledge Core object
   KnowledgeCore.configure(config).createClient(getContext())
     .onResult(new Async.ResultHandler<KnowledgeClient>() {

       @Override public void handleResult (Async<?> operation, @NonNull KnowledgeClient
   client) {
         // TO DO: Use the client instance
       }
     }
   );
   ```

   Note: If you're using enhanced domains, your org's Experience Cloud sites URL is different. For details, see My Domain URL Formats in Salesforce Help.

In Kotlin:

```
// Create a core configuration instance
val SITE_URL = "https://ExperienceCloudSitesSubdomainName.force.com"
val config = KnowledgeConfiguration.create(SITE_URL)

// Create a Knowledge Core object
KnowledgeCore.configure(config)
    .createClient(this.applicationContext)
    .onResult(object: Async.ResultHandler<KnowledgeClient> {

      override fun handleResult(operation: Async<*>?, result: KnowledgeClient) {
        // TO DO: Use the client instance
      }
    })
```

Note: You can get the site URL for the `create` method from your Salesforce org. If your Salesforce admin hasn't set up Knowledge in Service Cloud or you need more guidance, see Cloud Setup for Knowledge.

3. Fetch the article details as an `ArticleDetails` object from your knowledge base using the instructions in Article Fetching with the Knowledge Core API.

   You can fetch an article in several ways. The following example fetches an article using the unique article ID.

   In Java:

```
final String articleId = // Assign to the article ID

ArticleDetailRequest.builder(articleId)
  .cacheImages(true)
  .cacheResults(true)
  .submit(client)
  .onError(this)
  .onResult(new Async.ResultHandler<ArticleDetails>() {

    @Override
    public void handleResult (Async<?> async, @NonNull ArticleDetails articleDetails)
{
        // TO DO IN NEXT STEP: Show the article here
    }
});
```

   In Kotlin:

```
val articleId: String = // Assign to the article ID

ArticleDetailRequest.builder(articleId)
    .cacheImages(true)
    .cacheResults(true)
    .submit(client)
    .onError(this)
    .onResult(object : Async.ResultHandler<ArticleDetails> {

      override fun handleResult(async: Async<*>?, articleDetails: ArticleDetails) {
        // TO DO IN NEXT STEP: Show the article here
      }
    })
```

4. Display a web view containing the article using the `ArticleWebView` class.

   In Java:

```
// Configure the article web view
ArticleWebViewConfiguration articleConfig =
  new ArticleWebViewConfiguration.Builder(client)
    .setWebResourceErrorListener(new ArticleWebView.WebResourceErrorListener() {

      // Handle an error launching the article web view
      @Override public void onWebResourcesError (Uri uri, int errorCode, String
description) {
        // TO DO: Handle error
      }
}).build();

// Show the article
mArticleWebView.showArticle(articleDetails, articleConfig)
  .onComplete(new Async.CompletionHandler() {

    // Handle completion when showing article
    @Override public void handleComplete (Async<?> async) {
      // Make article web view visible
      mArticleWebView.setVisibility(View.VISIBLE);
    }
});
```

   In Kotlin:

```
// Configure the article web view
val articleConfig: ArticleWebViewConfiguration =
    ArticleWebViewConfiguration.Builder(client!!)
        .setWebResourceErrorListener(object: ArticleWebView.WebResourceErrorListener {

          // Handle an error launching the article web view
          override fun onWebResourcesError (uri: Uri, errorCode: Int, description:
String) {
            // TO DO: Handle error
          }
        }).build()

// Show the article
mArticleWebView.showArticle(articleDetails, articleConfig)
    .onComplete(object: Async.CompletionHandler {

      // Handle completion when showing article
      override fun handleComplete(async: Async<*>?) {
        // Make article web view visible
        mArticleWebView.setVisibility(View.VISIBLE)
      }
    })
```

Example: This example creates an Activity class that displays a Knowledge article using a specific article ID.

```
public class WebviewActivity extends AppCompatActivity implements Async.ErrorHandler
{

  // TO DO: Replace with a real site URL
  String siteUrl = "https://ExperienceCloudSitesSubdomainName.force.com";

  // TO DO: Replace with a real article ID
  String articleID = "kA0P00FAKE004SzKAI";

  // Article web view instance
  ArticleWebView mArticleWebView = null;

  /**
   * onCreate override.
   */
  @Override
  protected void onCreate (Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // TO DO: Update resource ID as necessary...
    setContentView(R.layout.activity_webview);
    mArticleWebView = findViewById(R.id.article_webview);

    // Initiate the process of showing the article
    launchAndShowKnowledge();
  }

  /**
   * Configures and launches the Knowledge interface.
   */
  void launchAndShowKnowledge() {

    // Create knowledge config object
    KnowledgeConfiguration config = KnowledgeConfiguration.create(siteUrl);

    // Create knowledge client object
    KnowledgeCore.configure(config).createClient(this)
        .onResult(new Async.ResultHandler<KnowledgeClient>() {
                    @Override public void handleResult (Async<?> operation,
                      @NonNull KnowledgeClient client) {

                        // Fetch the article based on the article ID
                        fetchArticle(articleID, client);
                    }
    });
  }

  /**
   * Fetches the Knowledge article.
   */
  void fetchArticle (final String articleId, final KnowledgeClient client) {
    // Request the article details
    ArticleDetailRequest.builder(articleId)
        .cacheImages(true)
        .cacheResults(true)
        .submit(client)
        .onError(this)
        .onResult(new Async.ResultHandler<ArticleDetails>() {
          @Override
          public void handleResult (Async<?> async,
            @NonNull ArticleDetails articleDetails) {

              // Use the article details to display the article
              displayArticleWebView(client, articleDetails);
          }
    });
  }

  /**
   * Displays the Knowledge article.
   */
  void displayArticleWebView (KnowledgeClient client, ArticleDetails articleDetails)
{

    // Create an article webview config object
    ArticleWebViewConfiguration configuration =
      new ArticleWebViewConfiguration.Builder(client)
      .setWebResourceErrorListener(new ArticleWebView.WebResourceErrorListener() {
        @Override public void onWebResourcesError (Uri uri, int errorCode, String
description) {
          // TO DO: Handle error event
        }
    }).build();

    // Show the article!
    mArticleWebView.showArticle(articleDetails, configuration)
      .onComplete(new Async.CompletionHandler() {
        @Override public void handleComplete (Async<?> async) {
          // TO DO: Handle complete event
        }
    });
  }

  /**
   * onDestroy override.
   */
  @Override protected void onDestroy () {
    super.onDestroy();
    if (mArticleWebView != null) {
      mArticleWebView.cleanup();
    }
  }

  /**
   * Async.ErrorHandler handleError override.
   */
  @Override public void handleError (Async<?> async, @NonNull Throwable throwable) {
    // TO DO: Handle error
  }
}
```

Using Case Management with the Service SDK

Add the Case Management experience to your mobile app.

Case Management in the Service SDK for Android

The Case Management feature in the SDK allows your users to create and manage cases.

Quick Setup: Case Publisher as a Guest User

To set up Case Publisher in your Android app, create a configuration object that points to your Experience Cloud site and then create a Case UI client.

Case Management as an Authenticated User

To manage existing cases, a user must authenticate with your org. Once authenticated, the user can create and manage cases from your app.

Cache and Encryption Behavior for Case Management

Case Management content is cached when it's fetched for an authenticated user. The cached data is encrypted using AES-256 encryption. You can explicitly clear cached content. No data is cached for unauthenticated users.

Send Custom Data Using Hidden Fields

By default, all fields specified in your global action layout are shown to users so that they can fill them in. In some circumstances, you might want to fill in a field without showing it to users.

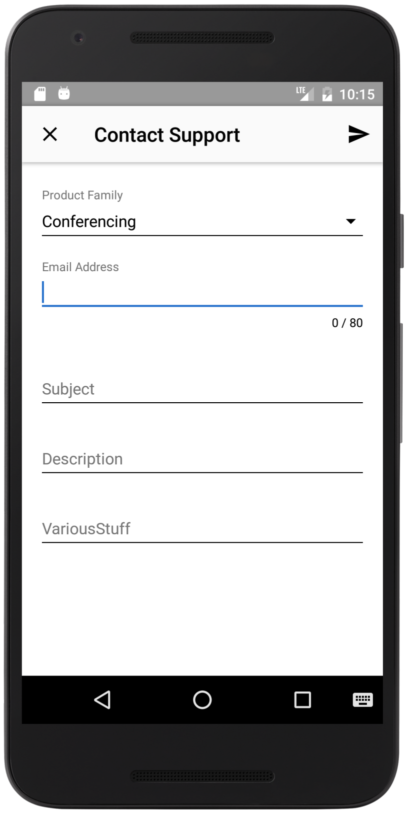
Push Notifications for Case Activity

You can send push notifications from your org when activity associated with a user's case occurs. When your app receives information from your org, pass the case ID to the SDK so the SDK can update the UI.

Case Management in the Service SDK for Android

The Case Management feature in the SDK allows your users to create and manage cases.

Once you point your app to your Experience Cloud site URL, you can display the case publisher interface to your users.



If you authenticate the user, you can display the user's list of cases and allow them to comment on any of their cases.

You can also customize the look and feel of the interface so that it fits naturally within your app. These customizations include the ability to fine-tune the colors, the fonts, the images, and the strings used throughout the interface.

Quick Setup: Case Publisher as a Guest User

To set up Case Publisher in your Android app, create a configuration object that points to your Experience Cloud site and then create a Case UI client.

Before starting, make sure that you've already:

- Set up Service Cloud to work with Case Management. See Cloud Setup for Case Management.
- Installed the SDK. See Install the Service SDK for Android.

These instructions set up Case Publisher as a guest user. When you activate the interface for a guest user, a generic Case Publisher screen appears.

1. Specify your Experience Cloud site URL and the global action to use to determine the fields shown when a user creates a case.

   The global action is used as the layout when creating a case.

   In Java:

   ```
   // Specify the site url and global action
   public static final String SITE_URL = "https://your-site-url";
   public static final String CREATE_CASE_ACTION_NAME = "NewCase";
   ```

In Kotlin:

```
// Specify the site url and global action
val SITE_URL = "https://your-site-url"
val CREATE_CASE_ACTION_NAME = "NewCase"
```

Note: You can get the required parameters from your Salesforce org.

2. (Optional) To send hidden fields, implement the `CaseClientCallbacks` interface and handle the `getHiddenFields` method.

   By default, all fields specified in your global action layout are shown to users so that they can fill them in. In some circumstances, you might want to fill in a field without showing it to users.

3. Create a `CaseConfiguration` object using the Experience Cloud site URL and the global action.

   In Java:

```
// Create a core configuration instance
CaseConfiguration coreConfiguration =
  new CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
                       .build();
```

   In Kotlin:

```
// Create a core configuration instance
val coreConfiguration =
  CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
                       .build()
```

   If you've created a callback to pass along hidden fields, build the configuration object with your callback function.

   In Java:

```
// Create configuration callback function
CaseClientCallbacks myCallback = new CaseClientCallbacks() {
  @Override public Map<String, String> getHiddenFields() {
    HashMap<String, String> hiddenFields = new HashMap<>();
    hiddenFields.put("CustomDescription__c", "This is a hidden field!");
    return hiddenFields;
  }
};

// Create a core configuration instance with callback
CaseConfiguration coreConfiguration =
  new CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
        .callbacks(myCallback) // Optional callback function
        .build();
```

   In Kotlin:

```
// Create configuration callback function
val myCallback = CaseClientCallbacks {
  val hiddenFields = HashMap<String,String>()
  hiddenFields.put("CustomDescription__c", "This is a hidden field!")
  hiddenFields
}

// Create a core configuration instance with callback
```

```
val coreConfiguration =
      CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
          .callbacks(myCallback) // Optional callback function
          .build()
```

4. Configure `CaseUI` using `CaseUIConfiguration`.

   In Java:

   ```
   // Create a UI configuration instance from a core instance
   CaseUI.with(MainActivity.this).configure(CaseUIConfiguration.create(coreConfiguration));
   ```

   In Kotlin:

   ```
   // Create a UI configuration instance from a core instance
   CaseUI.with(this@MainActivity).configure(CaseUIConfiguration.create(coreConfiguration))
   ```

5. (Optional) Customize the interface.

   You can customize the colors, strings, and other aspects of the interface. You can also localize the strings into other languages.

6. To display the Case Publisher interface to your user, launch the Case Publisher UI by calling the `launch` method.

   In Java:

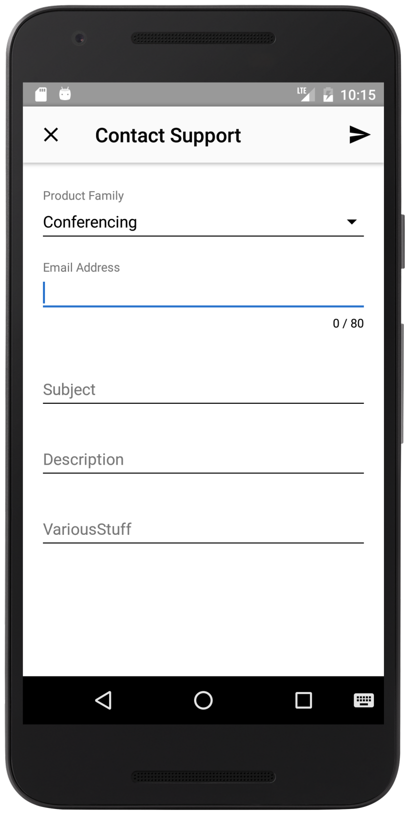
   ```
   // Create a client UI asynchronously
   CaseUI.with(MainActivity.this).uiClient()
                                 .onResult(new Async.ResultHandler<CaseUIClient>() {
     @Override public void handleResult(Async<?> async,
                                        @NonNull CaseUIClient caseUIClient) {
       caseUIClient.launch(MainActivity.this);
     }
   });
   ```

   In Kotlin:

   ```
   // Create a client UI asynchronously
   CaseUI.with(this@MainActivity).uiClient()
     .onResult { async, caseUIClient -> caseUIClient.launch(this@MainActivity) }
   ```

   Note: If a user isn't logged in, the `launch` method launches the Case Publisher activity. If a user is logged in, the Case List activity appears. To explicitly launch the Case Publisher activity regardless of whether a user is logged in, use `launchCasePublisher`.

After you launch the UI, the Case Publisher screen appears.



Case Management as an Authenticated User

To manage existing cases, a user must authenticate with your org. Once authenticated, the user can create and manage cases from your app.

Before starting, make sure that you've already:

- Set up Service Cloud to work with Case Management. See Cloud Setup for Case Management.
- Installed the SDK. See Install the Service SDK for Android.

These instructions allow you to set up case management as an authenticated user. When you activate the Case Management interface for an authenticated user, a list of their existing cases appears initially. From there, they can inspect an existing case, or they can create a new case. If you do not want to authenticate users, and you prefer to let them create cases as a guest user, see Quick Setup: Case Publisher as a Guest User.

1. Follow authentication instructions from Authentication with the Service SDK for Android.
2. Set up Case Management as described in Quick Setup: Case Publisher as a Guest User.

   Note: If you don't have an Experience Cloud site URL, you can use the instance URL you get back during the authentication process in place of the site URL during configuration.

   There are additional aspects to the setup process when dealing with authenticated users.

   a. In addition to the Experience Cloud site URL and the global action, specify a case list name.

      In Java:

```
// Specify the Experience Cloud site url, global action, and case list name
public static final String SITE_URL = "https://your-site-url";
public static final String CREATE_CASE_ACTION_NAME = "NewCase";
public static final String CASE_LIST_NAME = "MyCases";
```

In Kotlin:

```
// Specify the site url, global action, and case list name
val SITE_URL = "https://your-site-url"
val CREATE_CASE_ACTION_NAME = "NewCase"
val CASE_LIST_NAME = "MyCases"
```

To get this case list value, access the **Cases** tab in your org, pick the desired **View**, select **Go!** to see that view, and then select **Edit** to edit the view. From the edit window, you can see the **View Unique Name**.

Note: You can get the required parameters from your Salesforce org.

b. (Optional) To set up push notifications, see Push Notifications for Case Activity.

c. Build `CaseConfiguration` using the `caseListName` method and the `withAuthConfig` method.

In Java:

```
CaseConfiguration coreConfiguration =
  new CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
    .caseListName(CASE_LIST_NAME)
    .withAuthConfig(myAuthProvider, myAuthUser)
    .callbacks(myCallback) // Optional callback
    .build();
```

In Kotlin:

```
val coreConfiguration =
  CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
    .caseListName(CASE_LIST_NAME)
    .withAuthConfig(myAuthProvider, myAuthUser)
    .callbacks(myCallback) // Optional callback
    .build()
```

Warning: If you don't specify a case list value for `caseListName`, you'll get an `IllegalArgumentException` when configuring Case Management.

After you launch the UI as an authenticated user, the Case List screen appears.

Cache and Encryption Behavior for Case Management

Case Management content is cached when it's fetched for an authenticated user. The cached data is encrypted using AES-256 encryption. You can explicitly clear cached content. No data is cached for unauthenticated users.

To delete locally cached content, call the `clearCache` method from the `CaseClient` object. You can get this object using the `getCoreClient` method from `CaseUIClient`.

```
caseUIClient.getCoreClient().clearCache();
```

Send Custom Data Using Hidden Fields

By default, all fields specified in your global action layout are shown to users so that they can fill them in. In some circumstances, you might want to fill in a field without showing it to users.

To send hidden field data, create a `CaseClientCallbacks` object that returns the `getHiddenFields` method with a map of the hidden fields and their values.

In Java:

```
// Create configuration callback function
CaseClientCallbacks myCallback = new CaseClientCallbacks() {

  @Override public Map<String, String> getHiddenFields() {
    HashMap<String, String> hiddenFields = new HashMap<>();
    hiddenFields.put("FIELD_NAME__c", "FIELD_VALUE");
    return hiddenFields;
  }
};
```

In Kotlin:

```
// Create configuration callback function
var myCallback = object: CaseClientCallbacks {

  override fun getHiddenFields(): MutableMap<String, String> {
    var hiddenFields: HashMap<String, String> = HashMap()
    hiddenFields.put("FIELD_NAME__c", "FIELD_NAME")
    return hiddenFields
  }
}
```

When you build the configuration object (as described in Quick Setup: Case Publisher as a Guest User), specify your callback function.

In Java:

```
// Create configuration callback function
CaseClientCallbacks myCallback = new CaseClientCallbacks() {
  @Override public Map<String, String> getHiddenFields() {
    HashMap<String, String> hiddenFields = new HashMap<>();
    hiddenFields.put("CustomDescription__c", "This is a hidden field!");
    return hiddenFields;
  }
};

// Create a core configuration instance with callback
CaseConfiguration coreConfiguration =
  new CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
        .callbacks(myCallback) // Optional callback function
        .build();
```

In Kotlin:

```
// Create configuration callback function
val myCallback = CaseClientCallbacks {
  val hiddenFields = HashMap<String,String>()
  hiddenFields.put("CustomDescription__c", "This is a hidden field!")
  hiddenFields
}

// Create a core configuration instance with callback
val coreConfiguration =
      CaseConfiguration.Builder(SITE_URL, CREATE_CASE_ACTION_NAME)
          .callbacks(myCallback) // Optional callback function
          .build()
```

Push Notifications for Case Activity

You can send push notifications from your org when activity associated with a user's case occurs. When your app receives information from your org, pass the case ID to the SDK so the SDK can update the UI.

When you notify the SDK about case activity (using the `CaseUIClient.notifyCaseUpdated` method), the UI updates based on this new information.

- If the user is looking at their list of cases, the UI informs them that the list is out of date.
- If the user is looking at a case feed for that case, the UI refreshes and downloads new comments.

If you want to perform other actions based on case activity, such as displaying a notification to the user, use the `onPushNotificationReceived` method in your `PushNotificationListener` implementation.

1. Set up your app for authenticated Case Management, as described in Case Management as an Authenticated User.
2. Set up remote push notifications, as described in Push Notifications with the Service SDK for Android.
3. Create an Apex trigger in your org to detect activity.

   For general information about Apex triggers, see Using Apex Triggers to Send Push Notifications in the Salesforce Mobile Push Notifications Implementation Guide.

   You can use the following sample Apex code as a starting point. This Chatter Feed Item trigger sends a notification when an agent creates a text post on a case.

```
// THIS APEX TRIGGER IS PROVIDED AS AN EXAMPLE. BE SURE TO REVIEW
// YOUR CODE BEFORE PUTTING ANYTHING INTO PRODUCTION.

trigger newCaseFeedItemNotification on FeedItem (after insert) {

  for(FeedItem feedItem : Trigger.new) {

    try {

      Schema.SObjectType objectType = feedItem.parentId.getSObjectType();

      if(feedItem.body == null ) {
        // Don't push if we have no body.
        break;
      }

      // Ensure Case type
      if (objectType == Case.sObjectType) {

        Case cs = [SELECT contactId, ownerId, caseNumber, subject
                   FROM Case
                   WHERE id = :feedItem.parentId];
        Set<String> users = new Set<String>();
        // Determine who created or inserted this feed item
        String commentedById = feedItem.CreatedById;
        if (commentedById == null) {
          commentedById = feedItem.InsertedById;
          if (commentedById == null) {
            commentedById = feedItem.LastEditById;
          }
        }

        // If the FeedItem was not created by the owner, send to the owner
        if (cs.ownerId != null && !cs.ownerId.equals(commentedById)) {

          // Ensure the user has access to the feed item before pushing
          List<UserRecordAccess> accessList = [SELECT HasReadAccess, RecordId
            FROM UserRecordAccess
            WHERE UserId = :cs.ownerId
            AND RecordId = :feedItem.Id LIMIT 1];
          if (accessList != null && !accessList.isEmpty()
                                 && accessList[0].HasReadAccess) {
            users.add(cs.ownerId);
          }
        }

       // If the FeedItem was not created by the contact on the case send to the contact

        if (cs.contactId != null && !cs.contactId.equals(commentedById)) {

          // Ensure the user has access to the feed item before pushing
          List<UserRecordAccess> accessList = [SELECT HasReadAccess, RecordId
            FROM UserRecordAccess
            WHERE UserId = :cs.contactId
            AND RecordId = :feedItem.Id LIMIT 1];
          if (accessList != null && !accessList.isEmpty()
                                 && accessList[0].HasReadAccess) {
            users.add(cs.contactId);
          }
        }

        // Assemble the necessary payload parameters for the mobile app.
        // Params are:
        // (<alert text>,<alert sound>,<badge count>,<free-form data>)
        // This example doesn't use badge count but does make use of free-form
        // data to pass the caseId in the notification.
        // The number of notifications that haven't been acted
        // upon by the intended recipient is best calculated
        // at the time of the push. This timing helps
        // ensure accuracy across multiple target devices.

        // If subject is not set, use '(No Subject)'
        String subject = cs.subject;
        if (subject == null) {
          subject = '(No Subject)';
        }
```

```
        String alertText = 'New comment added to case: ' + subject;

        // Add the caseId so we can handle the push notification within the app
        Map<String, Object> freeFormData = new Map<String, Object>();
        freeFormData.put('caseid', cs.id);

        // Add any other free form data here...

        // Create the payload
        Messaging.PushNotification msg = new Messaging.PushNotification();

        // Format for apple devices
        Map<String, Object> payload =
          Messaging.PushNotificationPayload.apple(alertText, '', null, freeFormData);

        // Add payload to the notification
        msg.setPayload(payload);

        // Needs to match your connected app name
        msg.send('YourConnectedAppName', users);
      }
    }
    catch (Exception e) {
      // Catch everything to ensure push failures do not prevent posts from succeeding.

      // TO DO: Add logging here to record errors or display an error message.
    }

  } // end of for loop
}
```

Note: If you want the SDK to handle notifications about case activity, your free-form data **must** contain the case ID, as shown in this code sample.

4. In your `PushNotificationListener` implementation, extract the case ID and pass it to the `CaseUIClient.notifyCaseUpdated` method.

   For example, in Java:

```
public class MyPushNotificationListener implements PushNotificationListener {

  @Override public void onPushNotificationReceived (RemoteMessage message) {

    // Extract the Case ID from the data. The name of the key depends
    // on how you bundled the freeform data in your Apex trigger...
    String caseId = message.getData().get("caseid");

    // TO DO: Extract any other info from the data

    CaseUIClient uiClient = // TO DO: Get the UI client from configuration

    if (uiClient != null) {
      // Pass case update information to the UI client.
      uiClient.notifyCaseUpdated(caseId);
    }
  }
}
```

In Kotlin:

```
class MyPushNotificationListener : PushNotificationListener {

  override fun onPushNotificationReceived (message: RemoteMessage?) {

    // Extract the Case ID from the data. The name of the key depends
    // on how you bundled the freeform data in your Apex trigger...
    val caseId = message.getData().get("caseid")

    // TO DO: Extract any other info from the data

    var uiClient: CaseUIClient? = null

    // TO DO: Get the UI client from configuration

    if (uiClient != null) {
      // Pass case update information to the UI client.
      uiClient.notifyCaseUpdated(caseId)
    }
  }
}
```

Using SOS with the Service SDK

Add the SOS experience to your mobile app.

SOS in the Service SDK for Android
Learn about the SOS experience using the SDK.

Quick Setup: SOS in the Service SDK
To start an SOS session from your Android app, call the `SOS` class from an activity. Once the session is started, SOS automatically handles activity transitions for you.

Configure an SOS Session
Before starting an SOS session, you can optionally configure the session using an `SosConfiguration` object. This object allows you to enable or disable cameras, determine what screen a session starts on, and control other features.

Control an SOS Session
During an SOS session, you can control the session using various static methods on the `Sos` class. These methods can pause the session, disable screen sharing, stop the session, change what is being shown on the screen, and perform other functions.

Two-Way Video
In addition to screen sharing, the SOS SDK lets your customer share their device's live camera feed with an agent. The customer's front-facing camera allows for a video conversation with an agent. The back-facing camera provides a great way for a customer to show something to an agent, rather than have to explain it.

Check SOS Agent Availability

Before starting a session, you can check the availability of your SOS agents and then provide your users with more accurate expectations.

Listen to SOS Events

The SOS SDK provides listeners that allow you to listen and respond to various events emitted during an SOS session. These listeners can be used to keep your application informed about the state of the SOS session or to implement custom behavior based on the SOS session progress.

Quality-of-Service Events

Check your audio and video quality-of-service (QoS) to detect packet loss and other streaming issues between the OpenTok media router and your org.

Detect the Keyboard

Because the Android OS manages the keyboard, its representation on the screen is inaccessible to an application. This situation presents a challenge when sending an image of the device to an agent during an SOS session. For this reason, you should explicitly pass along information about the keyboard.

Field Masking

If an application contains sensitive information that an agent shouldn't see during an SOS session, you can hide this information from the agent.

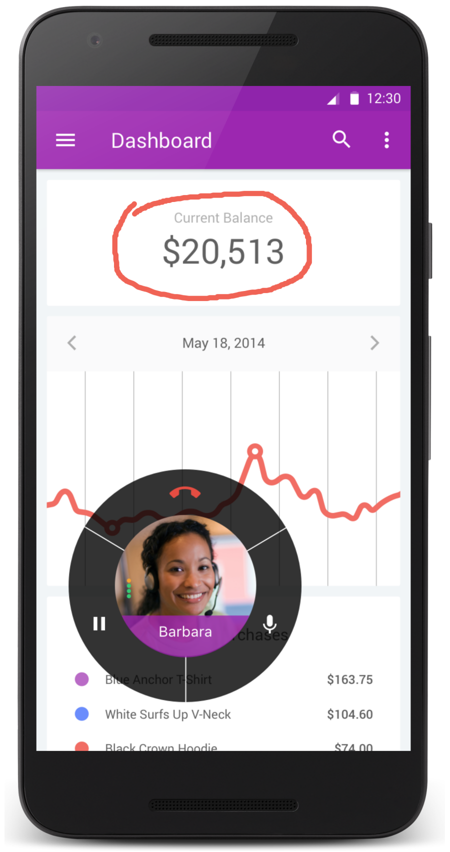
Custom Data

Use custom data to identify customers, send error messages, issue descriptions, or identify the page the SOS session was initiated from.

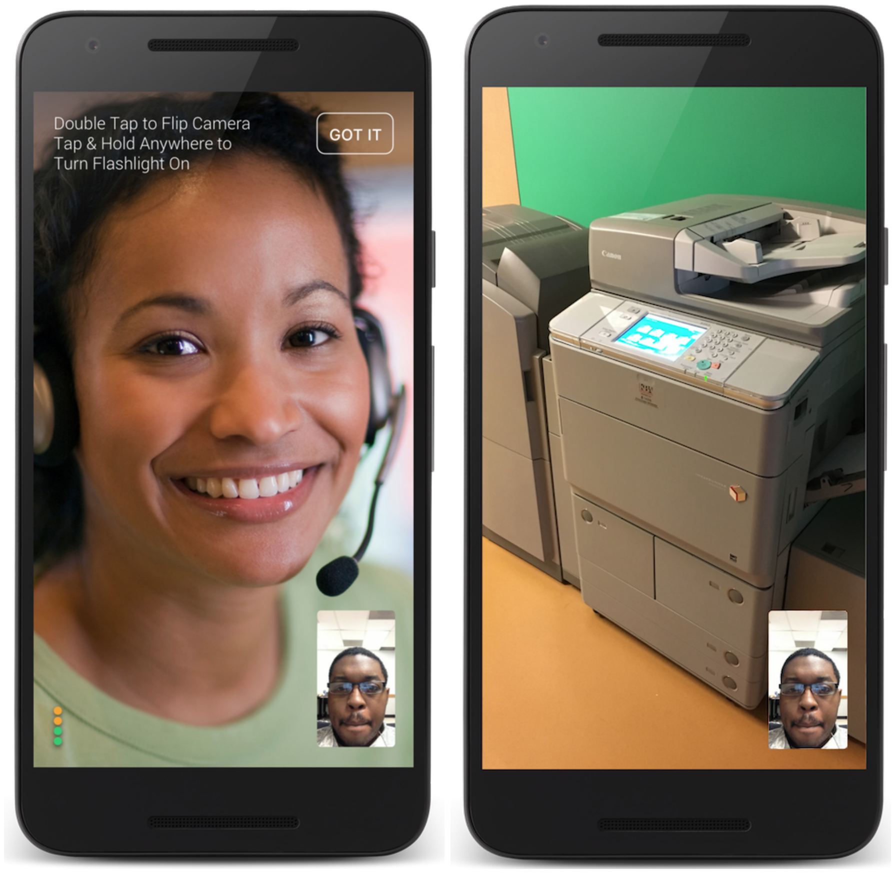
SOS in the Service SDK for Android

Learn about the SOS experience using the SDK.

SOS lets you easily add real-time video and screen sharing support to your native Android app. Once you've set up Service Cloud for SOS, it takes just a few calls to the SDK to have your app ready to handle agent calls and to support screen sharing. With screen sharing, agents can even make annotations directly on the customer's screen.



And with just a few more configuration changes, you can provide two-way video support from your app. This functionality can include front-facing camera support, back-facing camera support, or both.



There are several other ways you can set up your SOS environment, including masking sensitive fields and passing custom data back to your org. And if you want to handle state changes and other events, check out Listen to SOS Events.

Once you've configured SOS to work within your app, customize the look and feel of the interface so that it fits naturally within your app by using our UI Customization guidelines.

Let's get started.

Quick Setup: SOS in the Service SDK

To start an SOS session from your Android app, call the `SOS` class from an activity. Once the session is started, SOS automatically handles activity transitions for you.

Before running through these steps, be sure you've already:

- Set up Service Cloud to work with SOS. To learn more, see Console Setup for SOS.
- Installed the SDK. To learn more, see Install the Service SDK for Android.

Once you've reviewed these prerequisites, you're ready to begin.

1. Create an `Activity` subclass.

   Create a class that subclasses the `Activity` class.

2. From within a method of the `Activity` object, create an `SosOptions` instance with information about your LiveAgent pod, your Salesforce org ID, and the deployment ID.

In Java:

```
// Specify your org ID, your deployment ID, and your Chat endpoint
public static final String ORG_ID = "YOUR_ORG_ID";
public static final String DEPLOYMENT_ID = "YOUR_DEPLOYMENT_ID";
public static final String SOS_POD = "YOUR_ORG_URL";
                             // e.g. "d.la.salesforce.com"

// ...

// Create an SosOptions object
SosOptions options = new SosOptions(SOS_POD, ORG_ID, DEPLOYMENT_ID);
```

In Kotlin:

```
// Specify your org ID, your deployment ID, and your Chat endpoint
val ORG_ID = "YOUR_ORG_ID"
val DEPLOYMENT_ID = "YOUR_DEPLOYMENT_ID"
val SOS_POD = "YOUR_ORG_URL"
      // e.g. "d.la.salesforce.com"

// ...

// Create an SosOptions object
val options = SosOptions(SOS_POD, ORG_ID, DEPLOYMENT_ID)
```

Note: You can get the required parameters for this method from your Salesforce org. If your Salesforce admin hasn't already set up SOS in Service Cloud or you need more guidance, see Console Setup for SOS.

3. (Optional) Configure the SOS session before starting.

   Before starting an SOS session, you can optionally configure the session using an `SosConfiguration` object. This object allows you to enable or disable cameras, determine what screen a session starts on, and control other features. For more information about configuring a session, see Configure an SOS Session.

4. (Optional) Customize the interface.

   You can customize colors, strings, floating action buttons, and other aspects of the interface. You can also localize the strings into other languages. To learn more about customizations, see SDK Customizations with the Service SDK for Android.

5. Call `start` on a new SOS session using the `SosOptions` object created earlier.

   In Java:

```
// Start an SOS session
Sos.session(options).start(MainActivity.this);
```

   In Kotlin:

```
// Start an SOS session
Sos.session(options).start(this@MainActivity);
```

For additional details on customizing the SOS experience in your app, see the other topics covered in Using SOS with the Service SDK. If you run into network issues while connecting with an agent, see SOS Network Troubleshooting Guide.

Configure an SOS Session

Before starting an SOS session, you can optionally configure the session using an `SosConfiguration` object. This object allows you to enable or disable cameras, determine what screen a session starts on, and control other features.

`SosConfiguration` objects are constructed using `SosConfiguration.Builder`. For instance, the following code snippet enables a two-way video session:

```
// Enable two-way video in Java
SosConfiguration config = new SosConfiguration.Builder()
  .twoWayVideo(true)
  .build();

// Enable two-way video in Kotlin
val config = SosConfiguration.Builder()
  .twoWayVideo(true)
  .build()
```

Once you've built a configuration object, add it to the session that you're starting:

```
// Start session in Java
Sos.session(options)
  .configuration(config)
  .start(this);

// Start session in Kotlin
Sos.session(options)
  .configuration(config)
  .start(this)
```

You can use this configuration object for many different types of configuration settings.

Session Startup

By default there are several UIs that are presented to the user during the startup flow of an SOS session. This flow includes the device permissions UI, the onboarding UI, and the network test. Each of the sections of the startup flow can be disabled using the configuration builder.

Note: Disabling the permissions UI means that the application itself must acquire the necessary permissions before starting an SOS session.

To disable the permissions UI:

```
// Disable permissions UI in Java
SosConfiguration config = new SosConfiguration.Builder()
  .permissionUi(false)
  .build();

// Disable permissions UI in Kotlin
val config = SosConfiguration.Builder()
  .permissionUi(false)
  .build()
```

To disable the onboarding UI:

```
// Disable onboarding UI in Java
SosConfiguration config = new SosConfiguration.Builder()
  .onboardingUi(false)
  .build();

// Disable onboarding UI in Kotlin
val config = SosConfiguration.Builder()
  .onboardingUi(false)
  .build()
```

To change the layout of the onboarding UI:

```
// Change onboarding UI layout in Java
SosConfiguration config = new SosConfiguration.Builder()
  .onboardingCardLayouts(R.layout.onboardingCardOne, R.layout.onboardingCardTwo)
  .build();

// Change onboarding UI layout in Kotlin
val config = SosConfiguration.Builder()
  .onboardingCardLayouts(R.layout.onboardingCardOne, R.layout.onboardingCardTwo)
  .build()
```

To disable network tests:

```
// Disable network tests in Java
SosConfiguration config = new SosConfiguration.Builder()
  .networkTestEnabled(false)
  .build();

// Disable network tests in Kotlin
val config = SosConfiguration.Builder()
  .networkTestEnabled(false)
  .build()
```

Screen Sharing and Camera Configuration

By default, an SOS session begins by sharing the user's device screen with an agent. But you can also share the output from the device's cameras with an agent by enabling two-way video.

To enable two-way video:

```
// Enable two-way video in Java
SosConfiguration config = new SosConfiguration.Builder()
  .twoWayVideo(true)
  .build();

// Enable two-way video in Kotlin
val config = SosConfiguration.Builder()
  .twoWayVideo(true)
  .build()
```

To enter into "field services" mode, where screen sharing is disabled and SOS exclusively uses two-way video, use the following configuration:

```
// Enable field services mode in Java
SosConfiguration config = new SosConfiguration.Builder()
  .fieldServices(true)
  .build();

// Enable field services mode in Kotlin
val config = SosConfiguration.Builder()
  .fieldServices(true)
  .build()
```

There are many other ways you can configure a two-way video session. For more info about two-way video, see Two-Way Video.

Fonts

Although most UI customization is done by overriding XML resources, changing the font that is used to display messages is done by specifying the font asset with `typefaceFromAsset`.

```
// Change the font in Java
SosConfiguration config = new SosConfiguration.Builder()
  .typefaceFromAsset("fonts/myTypeface.ttf")
  .build();

// Change the font in Kotlin
val config = SosConfiguration.Builder()
  .typefaceFromAsset("fonts/myTypeface.ttf")
  .build()
```

Connecting UI

By default, connection messages appear on the SOS interface when a session connects. You can override this behavior and present your own UI during the connection process. To present your own UI for the connection process, disable the default UI at configuration time with the `connectingUi` method.

```
// Disable default UI in Java
SosConfiguration config = new SosConfiguration.Builder()
  .connectingUi(false)
  .build();

// Disable default UI in Kotlin
val config = SosConfiguration.Builder()
  .connectingUi(false)
  .build()
```

Now you can handle state changes with your own UI. To learn more, see SOS: Customize the Connecting UI.

Sounds

By default, the user's default system notification sound is played when an SOS session is fully connected and the agent appears on the screen. It is possible to customize the sound by providing a URI, providing a resource ID for an embedded sound file, or by disabling sounds altogether.

To use an embedded sound file:

```
// Add embedded agent join sound in Java
SosConfiguration config = new SosConfiguration.Builder()
  .agentJoinSound(R.raw.custom_agent_join_sound)
  .build();

// Add embedded agent join sound in Kotlin
val config = SosConfiguration.Builder()
  .agentJoinSound(R.raw.custom_agent_join_sound)
  .build()
```

To use a URI:

```
// Add URI agent join sound in Java
SosConfiguration config = new SosConfiguration.Builder()
  .agentJoinSound(Uri.parse("content://media/external/audio/media/710"))
  .build();

// Add URI agent join sound in Kotlin
val config = SosConfiguration.Builder()
  .agentJoinSound(Uri.parse("content://media/external/audio/media/710"))
  .build()
```

If you use a URI, we recommend using local or content provider paths only; a network resource may negatively impact the user experience.

To disable sound:

```
// Disable sound in Java
SosConfiguration config = new SosConfiguration.Builder()
  .playSounds(false)
  .build();

// Disable sound in Kotlin
val config = SosConfiguration.Builder()
  .playSounds(false)
  .build()
```

Debugging

When you are debugging and testing your application, it is often convenient to turn off some features. For instance, developing locally can cause an audio feedback loop between a device and your computer. If you are not working on audio features, you can use `audio` to disable all audio in your test sessions:

```
// Disable all audio in Java
SosConfiguration config = new SosConfiguration.Builder()
  .audio(false) // No audio will ever be available in the session
  .build();

// Disable all audio in Kotlin
val config = SosConfiguration.Builder()
  .audio(false) // No audio will ever be available in the session
  .build()
```

Another level of audio and video disabling is available using `agentPublish`. When set to `false` there is no creation, configuration, or use of the `OpenTok` agent. This is useful when using automated testing suites that do not have audio or video capabilities.

```
// Disable audio/video in Java
SosConfiguration config = new SosConfiguration.Builder()
  .agentPublish(false) // OpenTok agent is never used
  .build();

// Disable audio/video in Kotlin
val config = SosConfiguration.Builder()
  .agentPublish(false) // OpenTok agent is never used
  .build()
```

Control an SOS Session

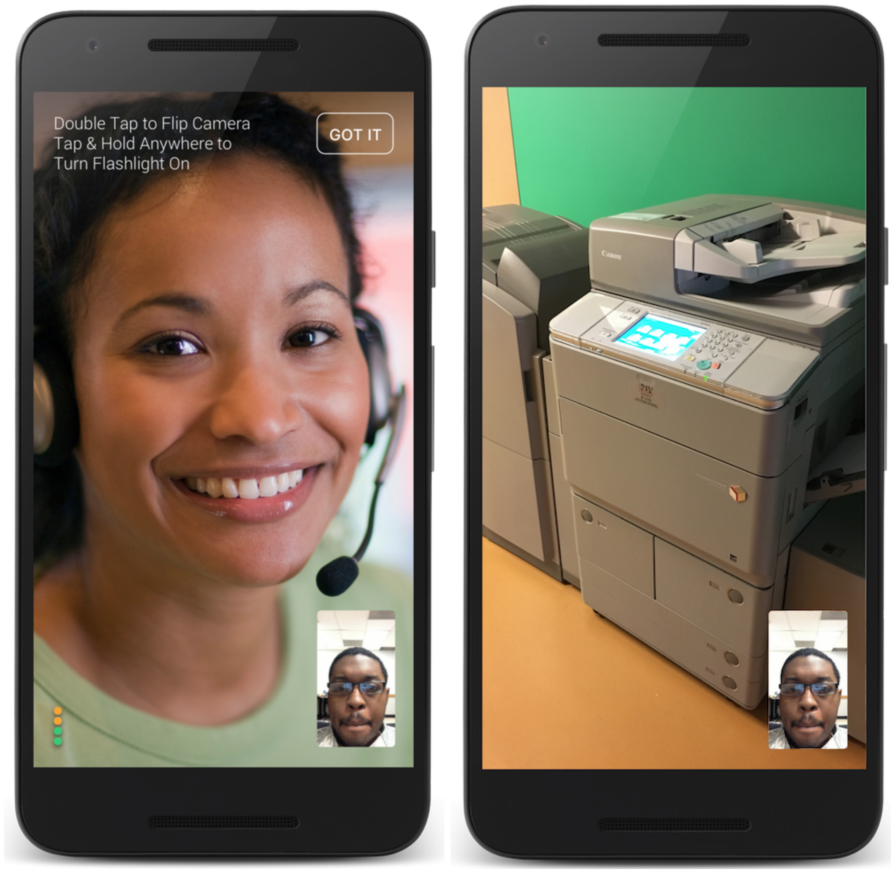
During an SOS session, you can control the session using various static methods on the `Sos` class. These methods can pause the session, disable screen sharing, stop the session, change what is being shown on the screen, and perform other functions.

Method Name	Description
`void setSessionPaused (boolean paused)`	Pauses or unpauses the session. When paused, no audio or video is sent to the agent.
`void endSession (boolean showPrompt)`	Ends the current SOS session. The `showPrompt` argument allows you to control whether a prompt is displayed to the user before the session ends.
`boolean isSessionActive ()`	Returns whether a session is active.
`void setScreenSharingEnabled (boolean enabled)`	Enables or disables the screen sharing functionality. This method can be used to temporarily stop screen sharing when the user is viewing sensitive information. See Field Masking for instructions on hiding specific fields.
`void setAudioEnabled (boolean enabled)`	Enables or disables the audio.
`void setTwoWayVideoEnabled (boolean enabled)`	Enables or disables the two-way video feature without prompting the user. You can only turn on two-way video if the session has been configured for two-way video and the device has a camera. See Two-Way Video to learn more.
`void setShareType (SosShareType shareType)`	Changes the share type to the specified value. Possible values are: `BackFacingCamera`, `FrontFacingCamera`, `ScreenSharing`. You can only turn on a camera if the session has been configured for two-way video and the device has a camera. See Two-Way Video to learn more.

Two-Way Video

In addition to screen sharing, the SOS SDK lets your customer share their device's live camera feed with an agent. The customer's front-facing camera allows for a video conversation with an agent. The back-facing camera provides a great way for a customer to show something to an agent, rather than have to explain it.

By default, after a connection is established, the camera shows the agent in the full-screen view and the customer's camera in the picture-in-picture view. If a device has both a front-facing and back-facing camera, the customer can swap cameras by double-tapping the screen during the two-way video session. The customer can also tap the picture-in-picture view to swap the full-screen view with the picture-in-picture view.



You can enable the camera features by creating an `SosConfiguration` object with the desired settings and adding this configuration object to the `Sos.SessionBuilder`. For details about using this mechanism, see Configure an SOS Session.

The following video-related configuration settings are available:

Table 5: Two-Way Video Configuration Settings

Setting	Default Value	Description
`twoWayVideo`	`false`	Whether two-way video is enabled. This setting is required if you want to use either of the cameras.
`frontFacingCamera`	`true`	Whether the front-facing camera is enabled. When two-way video is enabled, the front-facing camera is enabled by default, if available.
`backFacingCamera`	`true`	Whether the back-facing camera is enabled. When two-way video is enabled, the

Setting	Default Value	Description
		back-facing camera is enabled by default, if available.
`defaultShareType`	`SosShareType.ScreenSharing`	Default sharing mode when a session starts. Can be set to `SosShareType.ScreenSharing`, `SosShareType.FrontFacingCamera`, or `SosShareType.BackFacingCamera`. If you don't specify a default share type, when you open the camera, it will try to use the front-facing camera first. If you specify `FrontFacingCamera` or `BackFacingCamera`, be sure to enable `twoWayVideo` as well.
`fieldServices`	`false`	Whether the session is in field services mode. In field services mode, screen sharing is disabled and two-way video is enabled. The app launches in full-screen video mode and you can't go into screen sharing mode. By default, the front-facing camera is used (if available) when the session starts.

When you turn on the camera using SOS, the user sees the Camera UI. The SDK never sends camera frames to the agent without a clear indication for the user on the device.

The configuration process for some of the more common use cases are shown below.

Example: **Basic Two-Way Video**

To turn on basic two-way video:

```
// Turn on two-way video in Java
SosConfiguration config = new SosConfiguration.Builder()
  .twoWayVideo(true)
  .build();

// Turn on two-way video in Kotlin
val config = SosConfiguration.Builder()
  .twoWayVideo(true)
  .build()
```

In this scenario, two-way video is accessible to the user (by tapping the camera icon on the SOS UI), but the session still starts in screen sharing mode.

Example: **Start with Front-Facing Camera**

To start an SOS session with the front-facing camera:

```
// Start with front-facing camera in Java
SosConfiguration config = new SosConfiguration.Builder()
```

```
  .twoWayVideo(true)
  .defaultShareType(SosShareType.FrontFacingCamera)
  .build();

// Start with front-facing camera in Kotlin
val config = SosConfiguration.Builder()
  .twoWayVideo(true)
  .defaultShareType(SosShareType.FrontFacingCamera)
  .build()
```

In this scenario, the session starts with the front-facing camera, but the user can return to screen sharing if desired.

Example: **Field Services Mode**

To start an SOS session in field services mode:

```
// Start in field services mode in Java
SosConfiguration config = new SosConfiguration.Builder()
  .fieldServices(true)
  .build();

// Start in field services mode in Kotlin
val config = SosConfiguration.Builder()
  .fieldServices(true)
  .build()
```

In this scenario, the session starts in full-screen video mode (using the front-facing camera first, if available) and screen sharing mode is disabled.

Example: **Disable a Camera**

To enable two-way video but disable the back-facing camera:

```
// Enable two-way, disable back-facing camera in Java
SosConfiguration config = new SosConfiguration.Builder()
  .twoWayVideo(true)
  .backFacingCamera(false)
  .build();

// Enable two-way, disable back-facing camera in Kotlin
val config = SosConfiguration.Builder()
  .twoWayVideo(true)
  .backFacingCamera(false)
  .build()
```

To enable two-way video but disable the front-facing camera:

```
// Enable two-way, disable front-facing camera in Java
SosConfiguration config = new SosConfiguration.Builder()
  .twoWayVideo(true)
  .frontFacingCamera(false)
  .build();

// Enable two-way, disable front-facing camera in Kotlin
val config = SosConfiguration.Builder()
  .twoWayVideo(true)
```

```
    .frontFacingCamera(false)
    .build()
```

Check SOS Agent Availability

Before starting a session, you can check the availability of your SOS agents and then provide your users with more accurate expectations.

You can subscribe as a listener to be notified whenever the agent availability state changes.

1. Implement the `SosAvailability.Listener` interface.

   In Java:

   ```
   public class MyActivity extends Activity implements SosAvailability.Listener {
   ```

   In Kotlin:

   ```
   public class MyActivity: Activity(), SosAvailability.Listener {
   ```

2. Register your object as a listener.

   In Java:

   ```
   SosAvailability.addListener(this);
   ```

   In Kotlin:

   ```
   SosAvailability.addListener(this)
   ```

3. Upon registration, check the current status of agent availability to get initial status information.

   In Java:

   ```
   SosAvailability.Status status = SosAvailability.getStatus();
   ```

   In Kotlin:

   ```
   val status = SosAvailability.getStatus()
   ```

4. Upon registration, start polling for state changes.

   In Java:

   ```
   if (!SosAvailability.isPolling()) {
     SosAvailability.startPolling(this, orgId, deploymentId, liveAgentPod);
   }
   ```

   In Kotlin:

   ```
   if (!SosAvailability.isPolling()) {
     SosAvailability.startPolling(this, orgId, deploymentId, liveAgentPod)
   }
   ```

   Note: You can get the required parameters for this method from your Salesforce org. If your Salesforce admin hasn't already set up SOS in Service Cloud or you need more guidance, see Console Setup for SOS.

5. Implement the `onSosAvailabilityChange` method to handle state changes.

In Java:

```
@Override
public void onSosAvailabilityChange (SosAvailabilty.Status status) {
  // React to status updates: UNKNOWN, AVAILABLE, or UNAVAILABLE
}
```

In Kotlin:

```
override fun onSosAvailabilityChange(status: SosAvailability.Status?) {
  // React to status updates: UNKNOWN, AVAILABLE, or UNAVAILABLE
}
```

6. When done, unregister listener and stop polling.

   In Java:

```
SosAvailability.removeListener(this);
SosAvailability.stopPolling();
```

   In Kotlin:

```
SosAvailability.removeListener(this)
SosAvailability.stopPolling()
```

Refer to the `SosAvailability` Javadoc for more details.

Example: The following example handles agent availability changes from an `Activity` class.

```
public class MyActivity extends Activity implements SosAvailability.Listener {

  @Override
  protected void onCreate (Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    SosAvailability.addListener(this);

    // check if there is any known status, react to it if so
    updateAvailability(SosAvailability.getStatus());

    if (!SosAvailability.isPolling()) {
      SosAvailability.startPolling(this, orgId, deploymentId, liveAgentPod);
    }
  }

  @Override
  public void onDestroy () {
    super.onDestroy();

    SosAvailability.removeListener(this);

    // Stop polling, if needed.  Don't do this if you would like to continue
    // polling in the background
    SosAvailability.stopPolling();
  }

  //---------------------------------------------------------------------------
  // SosAvailability.Listener implementation
  //----------------------------------------------------------------------------

  @Override
  public void onSosAvailabilityChange (SosAvailabilty.Status status) {
    // React to status updates, either UNKNOWN, AVAILABLE, or UNAVAILABLE
    // This value is an enum

    // A primary use-case is to show a button to initiate an SOS session only if
    // there is an agent AVAILABLE
  }

  //----------------------------------------------------------------------------
  // private helpers
  //----------------------------------------------------------------------------

  private void updateAvailablity (SosAvailability.Status status) {
    // React to the given status, e.g. show a button if it is AVAILABLE
  }
}
```

Listen to SOS Events

The SOS SDK provides listeners that allow you to listen and respond to various events emitted during an SOS session. These listeners can be used to keep your application informed about the state of the SOS session or to implement custom behavior based on the SOS session progress.

Note: This topic covers how to handle state changes from your mobile app. For information about how to handle state changes from the Salesforce console, see Listen for SOS Console Events.

The following SOS listeners are available:

`SosListener`
Covers the high-level status of the session, including session creation, session termination, and the state of the session (with `SosState`).

`SosHoldListener`
Includes events for when the hold status of the session has changed.

`SosConnectionListener`
Allows you to listen for connection quality changes.

`SosAVListener`
Includes events for when the status of the audio or video playback changes during a session.

`SosMaskingListener`
Includes events for when masked fields are hidden or exposed during an SOS session.

`SosNetworkTestListener`
Can be used to listen to the status of the pre-session network test.

`SosShareTypeListener`
Allows you to monitor share type changes during a session.

`SosAvailability.Listener`
Allows you to monitor agent availability state changes. To learn more, see Check SOS Agent Availability.

`SosNetworkStatsListener`
Allows you to inspect your audio and video quality-of-service (QoS) to detect packet loss and other streaming issues. To learn more, see Quality-of-Service Events.

Refer to the linked Javadoc for more details about these listeners.

Checking Initial State

A few event types have static methods in order for you to manually check their states. Depending on your implementation, these methods can be used to check the state when you initially start listening for an event.

`getState`
Returns the same state sent to an `SosListener`.

`getHoldState`
Returns the same state sent to an `SosHoldListener`.

Using a Listener

Instances of each of the listener interfaces can be added or removed using the appropriate static APIs in the SOS class. Listeners can be added and removed at any time before a session starts or during an active session, and they are triggered for all events that occur until they're removed.

Note: Be sure to remove any listeners you've added when your application is finished listening to events. The added listeners are statically referenced, and failure to properly remove them before they go out of scope results in a memory leak in your application.

We recommend that you do not tie an `SosListener` implementation to the lifecycle of an `Activity` if you expect to receive SOS events during the session. Since Activities are created and destroyed with regularity, it is preferable to create an instance of `SosListener` that exists at the Application level, or by some other means to ensure it persists for the duration of a session.

Example: **Listener in Activity Class**

A common way to use the listener interfaces is to implement them in an `Activity` class. The `Activity` can add itself as a listener in one of its pre-visible lifecycle callbacks, and remove itself in a post-visible callback.

```
public class MyActivity extends Activity implements SosListener {

  // ...

  @Override
  public void onCreate (Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // ... Do your normal activity setup here.

    // Add this activity as an SOS event listener. The SosListener methods
    // implemented below will now be called for any new or existing SosSession
    // while this Activity instance exists.
    Sos.addListener(this);
  }

  @Override
  public void onDestroy () {
    super.onDestroy();
    // ... Do your normal activity teardown here.

    // Remove ourselves as an SOS event listener. This call ensures that you
    // will not leak a reference to your Activity.
    Sos.removeListener(this);
  }

  // ...

  @Override
  public void onSessionCreated () {
    // Handle a new session being created here by e.g. changing the UI to
    // reflect that you're in an SOS session.
  }

  @Override
  public void onSessionEnded (SosEndReason reason) {
    // Handle a session being ended here.
  }

  @Override
  public void onSessionStateChange (SosState state, SosState oldState) {
    // Handle state changes during the session here.
  }
}
```

Example: **Listener in Anonymous Class**

You can also choose to add a listener using an anonymous class for a more short-term use case. Be sure to remove the listener so there are no leaks.

```
public class MyActivity extends Activity {

  public static final String TAG = MyActivity.class.getSimpleName();

  private SosAVListener mListener;

  public void addListener () {
    mListener = new SosAVListener () {
      @Override
      public void onAudioToggled (boolean enabled) {
        LOG.d(TAG, "SOS audio is " + (enabled ? "enabled" : "disabled"));
      }

      @Override
      public void onVideoToggled (boolean enabled) {
        LOG.d(TAG, "SOS video is " + (enabled ? "enabled" : "disabled"));
      }
    };
  }

  public void removeListener () {
    if (mListener != null) {
      Sos.removeAVListener(mListener);
```

```
        mListener = null;
      }
    }

    @Override
    public void onDestroy () {
      // Make sure the listener is definitely removed.
      removeListener();
    }
}
```

Quality-of-Service Events

Check your audio and video quality-of-service (QoS) to detect packet loss and other streaming issues between the OpenTok media router and your org.

Note: This SDK allows you to track streaming issues on the other side of the conversation (from the agent to the media router). To track QoS issues on this side (from the app to the media router), see SOS Quality-of-Service Console Events.

1. Implement the `SosNetworkStatsListener` interface.

   This interface has 2 methods.

   ```
   void onAudioStatsChanged (AudioStats audioStats) {
     // Handle audio network stats updates
   }

   void onVideoStatsChanged (VideoStats videoStats) {
     // Handle video network stats updates
   }
   ```

   Each method provides streaming stats across a 30-second span of time.

   The `AudioStats` object gives you the following information:

   Table 6: AudioStats Methods

Method	Type	Description
getTimestamp	int	The timestamp for when the sample was taken, in milliseconds since the Unix Epoch.
getPacketsReceived	int	The total number of packets received on a stream since last update.
getPacketsLost	int	The total number of packets lost on a stream since last update.
getBytesReceived	int	The total number of bytes received on a stream since last update.
getSessionId	String	SOS Session ID that correlates to the data being provided.

   The `VideoStats` object gives you the following information:

Table 7: VideoStats Methods

Method	Type	Description
getTimestamp	int	The timestamp for when the sample was taken, in milliseconds since the Unix Epoch.
getPacketsReceived	int	The total number of packets received on a stream since last update.
getPacketsLost	int	The total number of packets lost on a stream since last update.
getBytesReceived	int	The total number of bytes received on a stream since last update.
getSessionId	String	SOS Session ID that correlates to the data being provided.
getSize	Size	The size of the agent video steam in pixels.

2. Register your object as a listener.

```
Sos.addNetworkStatsListener(myListener);
```

3. When done, unregister the listener.

```
Sos.removeNetworkStatsListener(myListener);
```

Detect the Keyboard

Because the Android OS manages the keyboard, its representation on the screen is inaccessible to an application. This situation presents a challenge when sending an image of the device to an agent during an SOS session. For this reason, you should explicitly pass along information about the keyboard.

Sending Keyboard Information

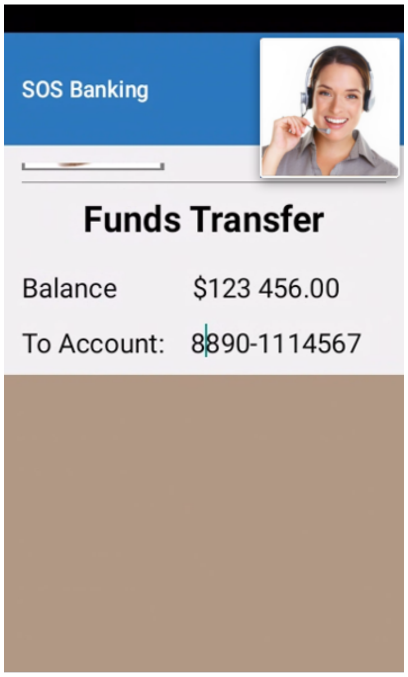
When you determine that keyboard information has changed, use the SDK to send this information to the agent:

`showKeyboard`
Informs the agent that the keyboard covers a given rectangle on the screen.

`hideKeyboard`
Informs the agent that the keyboard is not visible.

The API describes the size and position of the keyboard and not its appearance. Your application cannot know what the keyboard looks like, especially because users are free to install third-party keyboards. When you give the SDK a keyboard position and size the agent sees a shaded rectangle in the specified area. If you wish to show a full screen keyboard, pass in a `null Rect` to the `showKeyboard(Rect)` method.

Keyboard view from agent's perspective:



Techniques to Detect the Keyboard

There is no direct way to detect the keyboard on Android. However, it's possible to infer the existence of the keyboard by using the `InputMethodManager` to determine if input views are active on the screen and if they are able to receive input from the user.

It's also not possible to directly determine the position or dimensions of the keyboard, but sometimes you can infer this information by comparing the Window dimensions of the current `Activity` to the device's default `Display` area. In some situations, especially in portrait orientation, the `Window` dimensions shrink in height to accommodate the keyboard. If you subtract the shrunken `Window` dimensions from the `Display` dimensions, you can assume the keyboard occupies that area. In landscape orientation, if the `Window` size has not changed, you can assume that the keyboard takes up the entire display.

Detecting the Keyboard Using InputMethodManager

The `InputMethodManager` is provided by Android to facilitate the transfer of input data into an application. It also provides an API for determining if Android is currently accepting input. It doesn't provide any information about the size of the keyboard, but it's possible to infer whether a keyboard is displayed by combining the output from a few of its methods.

Once you have obtained an instance of the `InputMethodManager`, you can check for the presence of input-ready views with the following condition:

```
if (mInputMethodManager != null && mInputMethodManager.isActive() &&
  mInputMethodManager.isAcceptingText()) {
  // Determine keyboard area and call Sos.showKeyboard(...)
} else {
  Sos.hideKeyboard();
}
```

Invoking `InputMethodManager.isActive()` tells you if the current view has any editable views (such as `EditText`) inflated. Invoking `InputMethodManager.isAcceptingText()` tells you if any of those editable views have focus. It's possible for an editable view to have focus while the keyboard is not displayed. For this reason, it's not the only indicator you'll need to determine the on-screen presence of the keyboard.

The `InputMethodManager.isFullscreenMode()` method tells you if the keyboard is occupying the entire screen. In practice, this information isn't correlated with the appearance of a full screen keyboard, but rather that one is about to be displayed, or that the editable view that prompted the display of a full screen keyboard has retained focus. For this reason it's not a reliable indicator of when the full screen keyboard is displayed to the user.

Detecting the Keyboard Using Window Size

Another technique for detecting the keyboard is to obtain the current `Window` size. When a keyboard appears on the screen and doesn't require its own `Window`, as a full screen keyboard does, the current `Window` reduces its height to accommodate it. By subtracting the updated `Window` size from the default `Display` size, you can assume that the keyboard occupies the area in the difference. This technique works particularly well on tablets and in portrait orientation on smaller devices. However, this method does not work when a full screen keyboard is displayed (in landscape orientation on a phone) because the `Window` that contains your `View` does not change its dimensions.

There are other situations in which the `Window` size might change that are unrelated to the keyboard. For this reason, `Window` size is not the only indicator you'll need to determine the presence of the on-screen keyboard. We have found two methods for obtaining `Window` size changes that can help you detect a keyboard:

1. Intercept View Measurement
2. View Tree Observer Listeners

These two techniques only approximate the existence of the keyboard. Used together, they provide a reasonably accurate estimate. Taking into account the characteristics of your application, you can further improve the accuracy.

Window Size Method 1: Intercept View Measurement

In situations when a `Window` is resized to accommodate a keyboard, Android re-measures the views on screen using the Android `View` system. This process causes each layout's `onMeasure(int, int)` method to be called. An effective way to intercept this event is to find each root layout type you use for your activities and extend them to override the `onMeasure(int, int)` method. You should also change your layout XML files to use your new custom `Layout` classes.

```
package com.salesforce.documentation.layout;

public class CustomLinearLayout extends LinearLayout {

  // { Constructors Here }

  @Override
  protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    Rect rect = new Rect();
    ((Activity) getContext()).getWindow().getDecorView().
      getWindowVisibleDisplayFrame(rect);

    // Continue with Keyboard Detection, SOS API
  }
}
```

```
<com.salesforce.documentation.layout.CustomLinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/main_layout"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Views Here -->

</com.salesforce.documentation.layout.CustomLinearLayout>
```

Window Size Method 2: View Tree Observer Listeners

An alternative method to detect `Window` resizing is to use the `ViewTreeObserver`. Registering a `ViewTreeObserver.OnGlobalLayoutListener` with a `ViewTreeObserver` instance allows you to be notified when a view hierarchy is resized. This event can occur with the presence of an on-screen keyboard.

To obtain a reference to a `ViewTreeObserver`, you must obtain `Activity` instances as they are created. The most flexible way of getting this information is to register an instance of the `ActivityLifecycleCallbacks` with your `Application` context. Keep in mind that listeners you register with `ViewTreeObserver` should be explicitly cleaned up when the `Activity` it belongs to is paused or destroyed.

```
public class ActivityTracker implements Application.ActivityLifecycleCallbacks {

  private ViewTreeObserver mViewTreeObserver;
  private ViewTreeObserver.OnGlobalLayoutListener mGlobalLayoutListener;

  @Override
  public void onActivityResumed (final Activity activity) {
    View view = ((ViewGroup) activity.getWindow().getDecorView().
      findViewById(android.R.id.content)).getChildAt(0);
    mViewTreeObserver = view.getViewTreeObserver();
    mGlobalLayoutListener = new ViewTreeObserver.OnGlobalLayoutListener() {
      @Override
      public void onGlobalLayout () {
        Rect windowRect = new Rect();
        activity.getWindow().getDecorView().getWindowVisibleDisplayFrame(windowRect);
        // Continue with Keyboard Detection, SOS API
      }
    };

    mViewTreeObserver.addOnGlobalLayoutListener(mGlobalLayoutListener);
  }

  @Override
  public void onActivityPaused (Activity activity) {
    if (mViewTreeObserver == null || !mViewTreeObserver.isAlive() ||
        mGlobalLayoutListener == null) {
      return;
    }

    // Make sure the ViewTreeObserver and listener are cleaned up.
    if (android.os.Build.VERSION.SDK_INT <= 15) {
      mViewTreeObserver.removeGlobalOnLayoutListener(mGlobalLayoutListener);
    } else {
      mViewTreeObserver.removeOnGlobalLayoutListener(mGlobalLayoutListener);
    }
  }
  // { Other Overrides }
}
```

Example: **Keyboard Detector Example**

```
/**
 * KeyboardDetector
 *
 * Attempt to detect the presence of the soft keyboard on screen and prompt SOS
 * to display the keyboard overlay to the Agent.
 *
 * Utilizes the InputMethodManager to detect when a keyboard might be visible and
 * compares the Display size to the current Window size in order to determine what
 * the size of the keyboard could be.
 *
 * The InputMethodManager is able to tell us if there are any active input views
 * and if any of them are accepting input. We can use this to reasonably infer
 * that a soft keyboard could be present on the screen. Since we can only
 * correlate input readiness with the display of a soft keyboard, this method
 * is not 100% accurate.
 *
 * We are able to make an assumption regarding the soft keyboard's on-screen
 * dimensions expecting the Window dimensions to shrink once a keyboard is
 * displayed. By comparing the new, smaller Window dimensions with the
 * device's Display dimensions we can assume that the gap between the bottom of
 * the Window and the bottom of the Display must be the area occupied by a soft
 * keyboard. This method may not work if the Activity is configured to maintain
 * its dimensions, and it cannot account for the dimensions of the full-screen
 * keyboard that is displayed on phones while in landscape orientation.
 */
public class KeyboardDetector {

  private InputMethodManager mInputMethodManager;
  private WindowManager mWindowManager;
  private Activity mActivity;

  public KeyboardDetector (Activity activity) {
    mActivity = activity;
    mInputMethodManager =
      (InputMethodManager) activity.getSystemService(Context.INPUT_METHOD_SERVICE);
    mWindowManager =
      (WindowManager) activity.getSystemService(Context.WINDOW_SERVICE);
  }

  /**
   * Attempt to discover if a soft keyboard is displayed on the screen and what its
   * dimensions are. This method will inform SOS of when the keyboard overlay should
   * be hidden from the Agent and when it should be shown (and what its dimensions
   * are).
   */
  public void detectKeyboard () {
    if (mInputMethodManager == null ||
        !mInputMethodManager.isActive() ||
        !mInputMethodManager.isAcceptingText()) {
      Sos.hideKeyboard();
      return;
    }

    Rect windowRect = getWindowRect();

    if (mInputMethodManager.isFullscreenMode()) {
      Sos.showKeyboard(windowRect);
    } else {
      Rect displayRect = getDisplayRect();
      Rect keyboardRect = getKeyboardRect(displayRect, windowRect);
      Sos.showKeyboard(keyboardRect);
    }
  }

  /**
   * Get the dimensions of the Default Display as a Rect instance.
   *
   * @return Rect with the display dimensions
   */
  private Rect getDisplayRect () {
    Rect displayRect = new Rect();
    mWindowManager.getDefaultDisplay().getRectSize(displayRect);
    return displayRect;
  }

  /**
   * Get the dimensions of the Activity's Window as a Rect instance.
   *
   * @return Rect with the Window dimensions
   */
  private Rect getWindowRect () {
    Rect rect = new Rect();
    mActivity.getWindow().getDecorView().getWindowVisibleDisplayFrame(rect);
    return rect;
  }

  /**
   * Get the assumed dimensions of the soft keyboard by assuming that it must
   * occupy the area between the bottom of the Window Rect and the bottom of
   * the Display Rect.
   *
   * @param display Rect instance representing the Default Display area
   * @param window  Rect instance representing the Window area
   * @return Rect with assumed keyboard dimensions
   */
  private Rect getKeyboardRect (final Rect display, final Rect window) {
    return new Rect(display.left, window.bottom, display.right, display.bottom);
  }
}
```

Example: **Integration Example**One of the most reliable ways to invoke the `KeyboardDetector` is by extending the layout class you are using in your app and calling the `detectKeyboard()` method during `View.onMeasure(int, int)`. If you choose to invoke keyboard detection this way, you'll need to do this for each type of layout in your app that contains input views as they could cause a keyboard to appear.

If you choose to implement this on an `Activity` that will not be destroyed during an orientation change, you may want to invoke keyboard detection within an `onConfigurationChange(Configuration)` override as well.

```
public class KeyboardDetectingDrawerLayout
  extends android.support.v4.widget.DrawerLayout {

  private KeyboardDetector mKeyboardDetector;

  public KeyboardDetectingDrawerLayout (Context context) {
    this(context, null);
  }

  public KeyboardDetectingDrawerLayout (Context context, AttributeSet attrs) {
    this(context, attrs, 0);
  }

  public KeyboardDetectingDrawerLayout (Context context,
    AttributeSet attrs, int defStyle) {

    super(context, attrs, defStyle);
    mKeyboardDetector = new KeyboardDetector((Activity) context);
  }

  @Override
  protected void onMeasure (int widthMeasureSpec, int heightMeasureSpec) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec);
    mKeyboardDetector.detectKeyboard();
  }
}
```

Field Masking

If an application contains sensitive information that an agent shouldn't see during an SOS session, you can hide this information from the agent.

The Android SDK provides five views that mask information sent to an agent during an SOS session. These five views mimic their corresponding native Android views.

- `AutoCompleteTextView`
- `EditText`
- `MultiAutoCompleteTextView`
- `TextView`
- `View`

These views are used in the same way as their native Android counterparts, but they automatically obscure their contents when the application screen is being shared with an agent during an SOS session.

Implementing Masked Fields

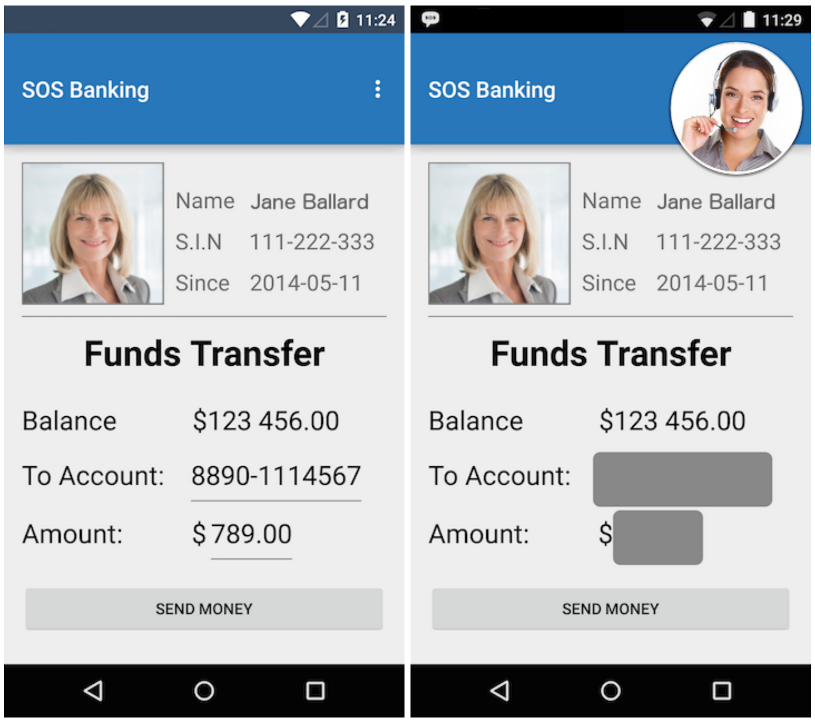
The simplest way to use field masking is to replace a native Android view in your layout file with its masking alternative.

In this example application that allows a user to transfer money to another account, the account number and the amount being transferred are sensitive data and are masked from the agent when in an SOS session. To get this behavior, instead of using a native Android `EditText` view for the account number and transfer amount, use `com.salesforce.android.sos.maskview.EditText`.

```
<com.salesforce.android.sos.maskview.EditText
    android:id="@+id/transfer_account"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF000000"
    android:text="@string/transfer_account_value"
    android:textSize="24sp"
    android:gravity="center_horizontal"/>

<com.salesforce.android.sos.maskview.EditText
    android:id="@+id/transfer_amount"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF000000"
    android:text="@string/transfer_amount_value"
    android:textSize="24sp"/>
```

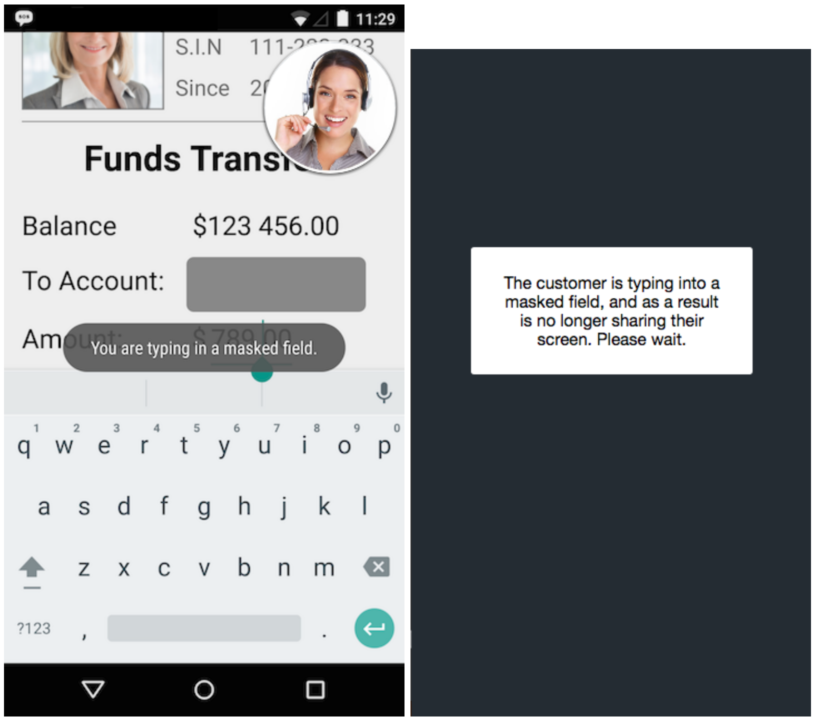
This change is enough to ensure that an agent doesn't see the values in these two views. When the user edits either of the fields, screen sharing halts and the agent sees a notification that sensitive data is being edited.

SOS session with masked field (normal view vs. SOS session view):



When a user begins editing a masked field, the view is exposed and a toast is presented to the user. Editing is then done normally as it would with the corresponding native Android view. The agent on the other end no longer sees the application screen and is presented with a notification.

Behavior when editing masked field (user view vs. agent view):


S.I.N
Since
Funds Trans
Balance $123 456.00
To Account:
You are typing in a masked field.
The customer is typing into a masked field, and as a result is no longer sharing their screen. Please wait.

When the user finishes editing the masked fields and the fields no longer have focus, screen sharing resumes with the fields masked and the agent able to see the application again.

Customizing Masked View Drawable

The default masking behavior is to draw a gray rounded rectangle to cover the masked view. You can customize the appearance of a masked field by specifying a `Drawable` resource to be used to draw the mask over the view. In this example application, we add a reference to a custom `Drawable` to each masked field to match the application's theme. This `Drawable` is used in place of the gray rounded rectangle when drawing.

Define a `Drawable` in `blue_field_mask.xml`:

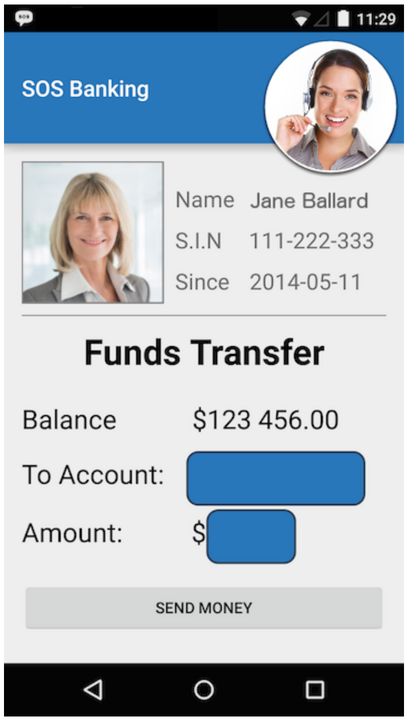
```
<?xml version="1.0" encoding="utf-8"?>
<shape
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">

    <!-- Specify a color for the background -->
    <solid
        android:color="#ff2877ba"/>

    <!-- Specify a darker border -->
    <stroke
        android:width="2dp"
        android:color="#ff36485d"/>

    <!-- Specify rounded corners -->
    <corners
        android:topLeftRadius="10dp"
        android:topRightRadius="10dp"
        android:bottomLeftRadius="10dp"
        android:bottomRightRadius="10dp"/>
</shape>
```

Add the custom `Drawable` to the masked views:

```
<com.salesforce.android.sos.mask.EditText
    app:sos_mask_drawable="@drawable/blue_field_mask"
    android:id="@+id/transfer_account"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF000000"
    android:text="@string/transfer_account_value"
    android:textSize="24sp"
    android:gravity="center_horizontal"/>

<com.salesforce.android.sos.mask.EditText
    app:sos_mask_drawable="@drawable/blue_field_mask"
    android:id="@+id/transfer_amount"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF000000"
    android:text="@string/transfer_amount_value"
    android:textSize="24sp"/>
```

As a result, the masks are now drawn using the custom `Drawable`. There is no restriction on what makes up the custom `Drawable`. The only caveat is that the `Drawable` stretches to fill the size of the masked view.

Custom Drawable masked field:



You can also change the `Drawable` used by a masked view programmatically by calling `setMask` on the masked view.

```
import com.salesforce.android.sos.mask.EditText;

...

Drawable blueMask = getResources().getDrawable(R.drawable.blue_field_mask);

final EditText transferAccountView = (EditText) rootView.findViewById(R.id.transfer_account);
transferAmountView.setMask(blueMask);

final EditText transferAmountView = (EditText) rootView.findViewById(R.id.transfer_amount);
transferAccountView.setMask(blueMask);
```

This method has the same effect as specifying a `Drawable` in XML, but it allows the mask to be set dynamically during application execution.

Manually Hiding Masked Fields

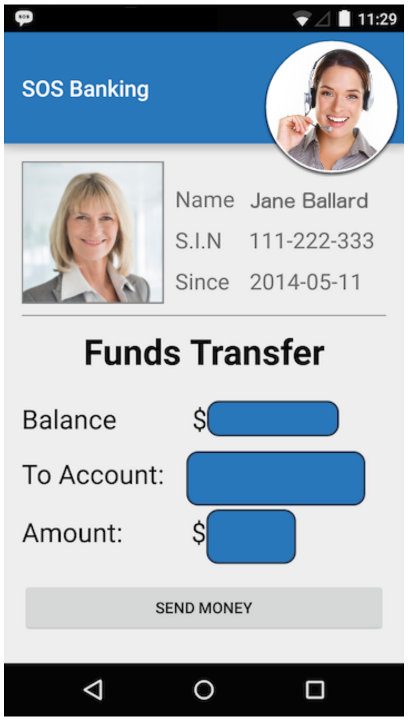
The default behavior for hiding and exposing masked fields is based on when a field has focus. If a masked field has focus, it is exposed (and screen sharing halts), and when it does not have focus, it is hidden. In situations where you want to expose and hide masked views manually, you can call `showMask` on the masked view. This method is useful in cases where a masked view does not normally get focus, such as displaying static text in a `TextView`, or when using the more generic `View`.

In this sample application, the account balance is considered sensitive information and is defined as a `TextView`.

```
<com.salesforce.android.sos.maskview.TextView
    app:sos_mask_drawable="@drawable/blue_field_mask"
    android:id="@+id/account_balance"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF000000"
    android:text="@string/account_balance_value"
    android:textSize="24sp"/>
```

```
import com.salesforce.android.sos.maskview.TextView;

...

final TextView accountBalance = (TextView) rootView.findViewById(R.id.account_balance);

// expose the account balance (do not mask the view)--screen sharing will be paused
accountBalance.showMask(false);

// hide the account balance (mask the view)
accountBalance.showMask(true);
```

Manually masked view:



In all situations where a masked field is exposed, regardless of whether it is exposed manually or automatically, the agent cannot view the application screen. Only when all masked fields are hidden is screen sharing enabled.

Turning Off Focus Masking

If you're planning on manually exposing and hiding a masked view, it can be confusing to have the default behavior based on focus active at the same time. This situation results in a visible masked view regardless of whether it has focus or if it has been manually exposed.

To turn off automatic masking based on focus, add a flag to the masked field definition:

```
<com.salesforce.android.sos.maskview.EditText
    app:sos_use_focus_masking="false"
    app:sos_mask_drawable="@drawable/blue_field_mask"
    android:id="@+id/transfer_account"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF000000"
    android:text="@string/transfer_account_value"
    android:textSize="24sp"
    android:gravity="center_horizontal"/>

<com.salesforce.android.sos.maskview.EditText
    app:sos_use_focus_masking="false"
    app:sos_mask_drawable="@drawable/blue_field_mask"
    android:id="@+id/transfer_amount"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textColor="#FF000000"
    android:text="@string/transfer_amount_value"
    android:textSize="24sp"/>
```

Now, when the user edits one of the masked fields, the mask is not removed. Make this change only if you plan to manage masking manually.

You can also turn off focus masking programmatically by calling `useFocusMasking` on the masked view.

```
import com.salesforce.android.sos.maskview.EditText;

...

final EditText transferAccountView = (EditText) rootView.findViewById(R.id.transfer_account);

// turn off focus masking
transferAmountView.useFocusMasking(false);

// turn focus masking back on
transferAmountView.useFocusMasking(true);
```

Custom Data

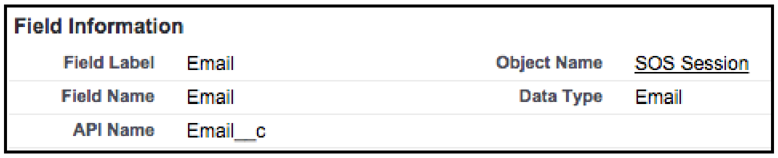
Use custom data to identify customers, send error messages, issue descriptions, or identify the page the SOS session was initiated from.

When an agent receives an SOS call, it can be helpful to have information about the caller before starting the session. Use the custom data feature to identify customers, send error messages, identify the currently viewed page, or send other information. Custom data populates custom fields on the SOS Session object that is created within your Salesforce org for each SOS session initiated by a user.

Before using custom data, create the corresponding fields within the SOS Session object of your Salesforce org. To learn more, see Create Custom Fields.

To use this feature, construct a `Map` instance and pass it to the `SosOptions` constructor. The keys in this map should reference the API Name for fields defined in your SOS Session object and the values should reflect the desired values for those fields. The class of the value object should reflect the field type in the SOS Session object.

Example: This example shows how to pass email information from your app to Service Cloud. Before trying this example, be sure to define an "Email" custom field on the SOS Session object in your Salesforce org:



Field Information

Field Label	Email	Object Name	SOS Session
Field Name	Email	Data Type	Email
API Name	Email__c		

To learn more about custom fields, see Create Custom Fields in Salesforce Help.

Once you have created a custom field, construct a `Map` and use the API Name of the field to specify the customer's email address. In this case, the field is defined as an Email data type, so we specify a valid email address as a `String`.

In Java:

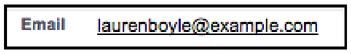
```
Map<String, Object> customData = new HashMap<>();

// Here we are passing the customer's email address as a String. Note the use
// of the field's API Name as the key in the map. We are only populating a single
// field here, but we may put an arbitrary number of entries into the map to
// populate multiple different fields.
customData.put("Email__c", "laurenboyle@example.com");
```

```
// Use the custom data map when initializing the SosOptions instance. From here,
// you may simply create the session as normal and the custom data will be used
// to populate fields in the SOS Session object.
SosOptions sosOpts = new SosOptions(
    customData
    "your.pod.name",
    "your-org-id",
    "your-deploy-id"
);
```

In Kotlin:

```
var customData = HashMap<String, Any>()

// Here we are passing the customer's email address as a String. Note the use
// of the field's API Name as the key in the map. We are only populating a single
// field here, but we may put an arbitrary number of entries into the map to
// populate multiple different fields.
customData.put("Email__c", "laurenboyle@example.com")

// Use the custom data map when initializing the SosOptions instance. From here,
// you may simply create the session as normal and the custom data will be used
// to populate fields in the SOS Session object.
val sosOpts = SosOptions(
    customData,
    "your.pod.name",
    "your-org-id",
    "your-deploy-id"
)
```

When the user creates an SOS session, the Email field is pre-populated with the value specified in the `SosOptions` custom data map.


Email laurenboyle@example.com

SDK Customizations with the Service SDK for Android

Once you've played around with some of the SDK features, use this section to learn how to customize the Service SDK so that it fits the look and feel of your app. This section also contains instructions for localizing strings in all supported languages.

Customize Colors with the Service SDK
You can customize the look and feel of the interface by specifying the colors used throughout the UI.

Customize and Localize Strings with the Service SDK
You can change the text throughout the user interface. To customize text, create string resource XML files (named `strings.xml`) in your project's `values-[locale]` resource folder for the language(s) you want to update.

Knowledge: Customize Action Buttons
You can add more views, such as floating action buttons, that appear in your Knowledge activities by implementing `KnowledgeViewAddition`. Adding views is handy if you want to give your users access to other features from your support home screen. Use this technique to add action buttons to Case Management, Chat, SOS, or any other features.

Knowledge: Customize Fonts

You can customize the fonts used in the Knowledge UI.

SOS: Customize the Connecting UI

By default, connection messages appear on the SOS interface when a session connects. You can override this behavior and present your own UI during the connection process.

SOS: Agent Annotations

During an SOS session, the agent can annotate a customer's screen to point out something . By default, agent annotations are red lines that are `5dp` wide, but you can customize these values.

SOS: Toast Behavior

The SOS session provides context to the customer through toasts for various events over the lifetime of the session. You can customize these toasts.

Customize Colors with the Service SDK

You can customize the look and feel of the interface by specifying the colors used throughout the UI.

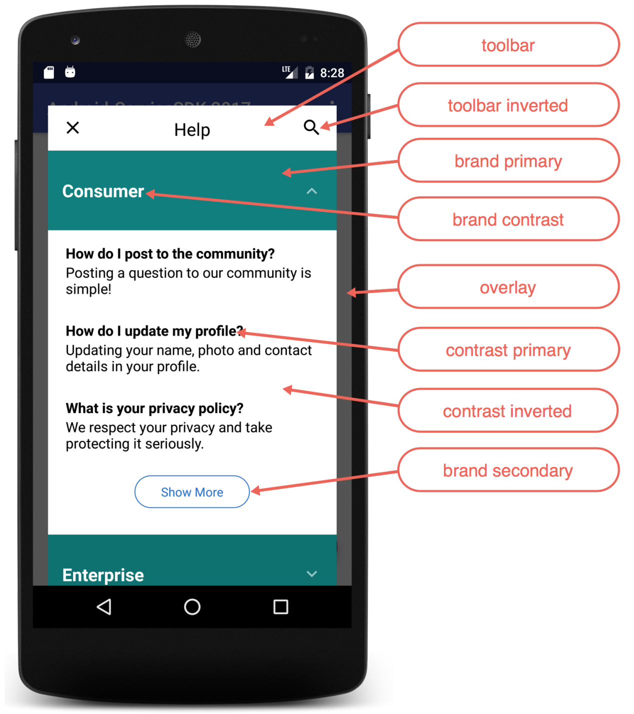
To customize the colors, create color resource values in your project's `colors.xml` file that correspond to the same resource names specified in this documentation.

For example, the following resource file values customize some of the branding tokens:

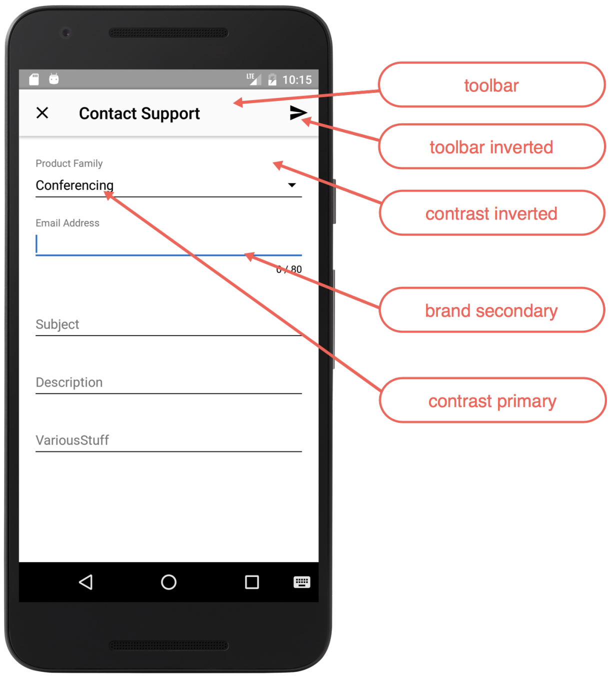
```
<resources>
  <color name="salesforce_brand_primary">#50e3c2</color>
  <color name="salesforce_brand_secondary">#4a90e2</color>
  <color name="salesforce_contrast_inverted">#ffffff</color>
  <color name="salesforce_contrast_primary">#333333</color>
  <color name="salesforce_contrast_secondary">#767676</color>
  <color name="salesforce_feedback_primary">#e74c3c</color>
</resources>
```

These screenshots illustrate how the branding tokens affect the UI.

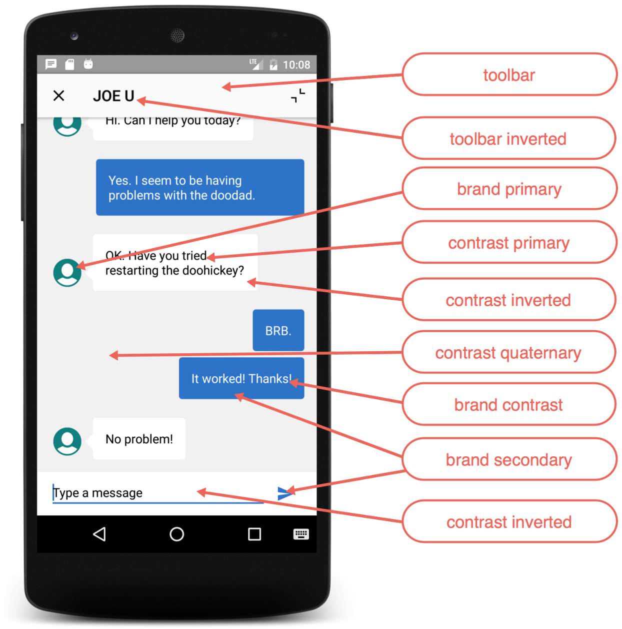
Knowledge UI Branding:


toolbar
toolbar inverted
brand primary
brand contrast
overlay
contrast primary
contrast inverted
brand secondary
8:28
Help
Consumer
How do I post to the community?
Posting a question to our community is simple!
How do I update my profile?
Updating your name, photo and contact details in your profile.
What is your privacy policy?
We respect your privacy and take protecting it seriously.
Show More
Enterprise

Case Management UI Branding:



Chat UI Branding:



Hi. Can I help you today?

Yes. I seem to be having problems with the doodad.

OK. Have you tried restarting the doohickey?

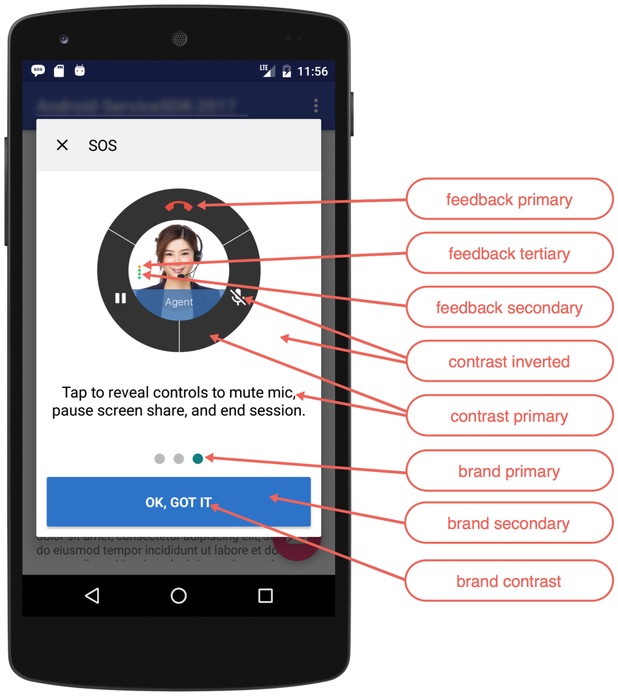
BRB.

It worked! Thanks!

No problem!

Type a message

SOS UI Branding:



Tap to reveal controls to mute mic, pause screen share, and end session.

The following branding tokens are available for customization.

Token Name	Default Value	Description / Sample Uses
Toolbar `salesforce_toolbar`	#FAFAFA ⬜	The toolbar background color.

Token Name	Default Value	Description / Sample Uses
Toolbar Inverted `salesforce_toolbar_inverted`	#010101 ⬛	Toolbar text and icon color.
Brand Primary `salesforce_brand_primary`	#007F7F 🟩	The banner background color. Knowledge: First data category, the Show More button, the footer stripe, the selected article. Chat: Line under the message you're typing. SOS: Used by various icons.
Brand Secondary `salesforce_brand_secondary`	#2872CC 🟦	Knowledge: UI button colors. Case Management: Action button background and the line under the currently selected field. Chat: User text bubbles. SOS: Background color for action items.
Brand Contrast `salesforce_brand_contrast`	#FCFCFC ⬜	Title text color. Knowledge: Text on data category headers and the chevron on the Knowledge home page.
Contrast Primary `salesforce_contrast_primary`	#000000 ⬛	Primary body text color. SOS: Background color for buttons on the UI.
Contrast Secondary `salesforce_contrast_secondary`	#6D6D6D ◼	Knowledge: Subcategory headers.
Contrast Tertiary `salesforce_contrast_tertiary`	#BABABA ▨	SOS: Dots in UI.
Contrast Quaternary `salesforce_contrast_quaternary`	#F1F1F1 □	Chat: Background color for the chat feed screen.

Token Name	Default Value	Description / Sample Uses
Contrast Inverted `salesforce_contrast_inverted`	#FFFFFF ⬜	Page background, navigation bar, table cell background. Chat: Color of the close button on the minimized view. Client message text. Background color at the bottom of the input bar on the chat feed UI. SOS: Colors of the icons (when they aren't selected).
Feedback Primary `salesforce_feedback_primary`	#E74C3C 🟥	Text color for error messages. SOS: Mute indicator. Disconnect icon.
Feedback Secondary `salesforce_feedback_secondary`	#2ECC71 🟩	SOS: Connection quality indicators. Background color for the Resume button when the two-way camera is active.
Feedback Tertiary `salesforce_feedback_tertiary`	#F5A623 🟧	SOS: Connection quality indicators.
Title Color `salesforce_title_color`	#FBFBFB ⬜	Text as it appears in titles throughout the UI. Text on areas where a brand color is used for the background. Chat: Agent text bubbles. SOS: Colors of the icons (when they are selected).
Overlay `salesforce_overlay`	#000000 (40% alpha)	Knowledge: Background for the Knowledge home screen.

Customize and Localize Strings with the Service SDK

You can change the text throughout the user interface. To customize text, create string resource XML files (named `strings.xml`) in your project's `values-[locale]` resource folder for the language(s) you want to update.

To see the complete list of string resource values, refer to the string resources document for the feature you want to customize.

- Chat String Resources
- Knowledge String Resources
- Case Management String Resources
- SOS String Resources

- Common String Resources

SDK text is translated into more than 25 different languages. In order for your string customizations to take effect in all languages, provide a translation for each language. To add support for a language, create a resources subdirectory that includes a hyphen and the ISO language code at the end of the directory name. For example, `values-es/` is the directory containing string resources for Spanish. Android loads the appropriate resources according to the locale settings of the device at run time. The system falls back on the strings in the default `values/` directory if the appropriate locale directory isn't found.

The following languages are currently supported:

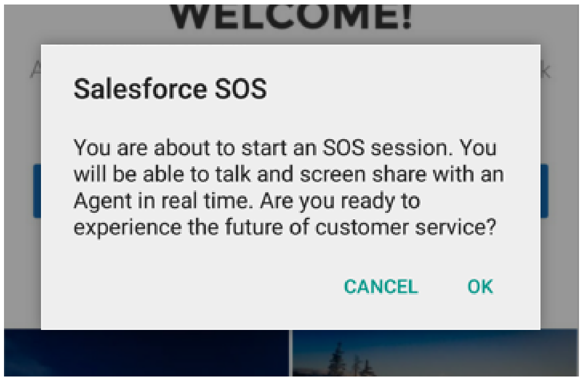
Table 8: Supported Languages

Language Code	Language
values-ar	Arabic
values-cs	Czech
values-da	Danish
values-de	German
values-el	Greek
values-en	English
values-es	Spanish
values-fi	Finnish
values-fr	French
values-hu	Hungarian
values-in	Indonesian
values-it	Italian
values-iw	Hebrew
values-ja	Japanese
values-ko	Korean
values-nl	Dutch
values-no	Norwegian
values-pl	Polish
values-pt-rBR	Brazilian Portuguese
values-ro	Romanian
values-ru	Russian
values-sv	Swedish
values-th	Thai
values-tr	Turkish

Language Code	Language
values-uk	Ukranian
values-vi	Vietnamese
values-zh	Chinese
values-zh-rTW	Traditional Chinese

Check out Supporting Different Languages in the Android Developer documentation for more info about localization.

Example: To learn how you can change string values, let's go through an example. The image below shows the default connection prompt dialog text in English:



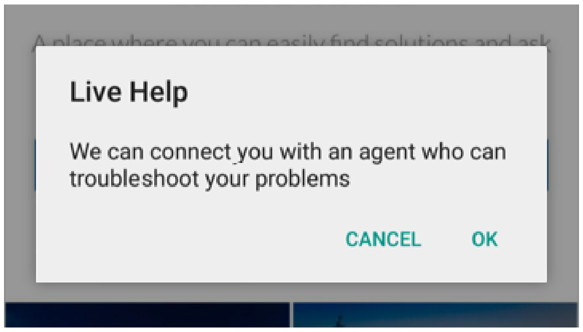
Salesforce SOS

You are about to start an SOS session. You will be able to talk and screen share with an Agent in real time. Are you ready to experience the future of customer service?

You can change the title and body of this dialog by changing the `sos_title` and the `sos_connect_prompt` strings in the `strings.xml` file in the `values` folders for your locale (`values-en/` for English):

```
<!-- other string resources omitted -->

<string name="sos_title">Live Help</string>
<string name="sos_connect_prompt">
  We can connect you with an agent who can troubleshoot your problems</string>
```

Now, whenever you start a session you see the updated dialog text:



Live Help

We can connect you with an agent who can troubleshoot your problems

Knowledge: Customize Action Buttons

You can add more views, such as floating action buttons, that appear in your Knowledge activities by implementing `KnowledgeViewAddition`. Adding views is handy if you want to give your users access to other features from your support home screen. Use this technique to add action buttons to Case Management, Chat, SOS, or any other features.

To learn more, see Add Knowledge View Additions.

Knowledge: Customize Fonts

You can customize the fonts used in the Knowledge UI.

By default, the Service SDK interface uses the OS default font. If you want to change the font, follow these steps.

1. Add a True Type Font (TTF) file to your project's `assets` directory.
2. Override the `SalesforceFontStyle` style in your project's `styles.xml` resource file and set the `salesforceFont` item to the relative asset path of your TTF file.

   For example:

   ```
   <style name="SalesforceFontStyle">
     <item name="salesforceFont">CustomFont.ttf</item>
   </style>
   ```

The Knowledge UI uses the selected font when you rebuild your app.

SOS: Customize the Connecting UI

By default, connection messages appear on the SOS interface when a session connects. You can override this behavior and present your own UI during the connection process.

To present your own UI for the connection process, disable the default UI at configuration time with the `connectingUi` method.

```
// Disable default UI in Java
SosConfiguration config = new SosConfiguration.Builder()
  .connectingUi(false)
  .build();

// Disable default UI in Kotlin
val config = SosConfiguration.Builder()
  .connectingUi(false)
  .build()
```

To learn more about using `SosConfiguration`, see Configure an SOS Session.

After you've disabled the default connecting UI, use the `SosListener` to listen for state changes and display your own UI. To learn more about listeners, see Listen to SOS Events.

Example: The following Fragment class listens to SOS state changes and provides a place for you to add your own UI.

```
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import com.salesforce.android.sos.api.Sos;
```

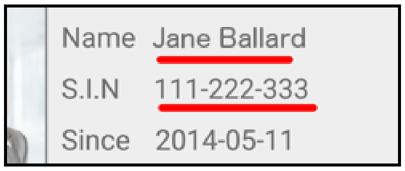
```
import com.salesforce.android.sos.api.SosListener;
import com.salesforce.android.sos.api.SosState;
import com.salesforce.android.sos.api.SosEndReason;

public class CustomConnectingUI extends Fragment implements SosListener {

  public CustomConnectingUI () {
  }

  public static CustomConnectingUI newInstance () {
    CustomConnectingUI fragment = new CustomConnectingUI();
    Bundle args = new Bundle();
    fragment.setArguments(args);
    return fragment;
  }

  @Override
  public View onCreateView (LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
    // Inflate the layout for this fragment
    View view = inflater.inflate(R.layout.fragment_custom_connecting_ui, container,
false);

    // Add an SosListener...
    Sos.addListener(this);

    // Set the initial state
    displayState(Sos.getState());

    return view;
  }

  @Override
  public void onDestroyView () {
    super.onDestroyView();

    // Remove listener
    Sos.removeListener(this);
  }

  @Override
  public void onSessionCreated () {
  }

  @Override
  public void onSessionEnded (SosEndReason reason) {
  }

  @Override
  public void onSessionStateChange (SosState state, SosState oldState) {
    displayState(state);
  }

  /**
```

```
   * Helper function to display SOS state changes.
   */
  private void displayState (SosState state) {
    if (state == SosState.Initializing) {
      // TO DO: Display a message that we're creating a session
    }
    if (state == SosState.AgentJoining) {
      // TO DO: Display a message that agent is joining
    }
    if (state == SosState.WaitingForAgent) {
      // TO DO: Display a message that we're waiting for agent to accept
    }
  }
}
```

SOS: Agent Annotations

During an SOS session, the agent can annotate a customer's screen to point out something . By default, agent annotations are red lines that are `5dp` wide, but you can customize these values.

By default, an annotation in SOS looks like this:



Name	Jane Ballard
S.I.N	111-222-333
Since	2014-05-11

The Android SDK offers customization of the line width and color. When you change the line width or color, the value you change it to syncs with the customer and the agent. To change the width or color of the annotation line, modify two XML resources: `sos_drawing_color` and `sos_drawing_width`.

Table 9: Agent Annotation Style

Annotation Name	Type	Default Value
`sos_drawing_color`	Color	#ffff0000
`sos_drawing_width`	Integer	5

For documentation about overriding resources in Android, see Resource Merging in the Android documentation.

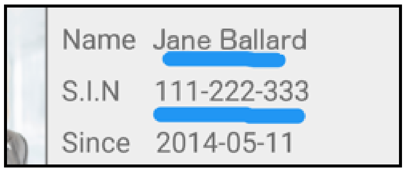
Example: Let's say we want to change the annotation to a thicker blue line. Specify the color value for `sos_drawing_color` (alongside your other resources):

```
<color name="sos_drawing_color">#ff2196f3</color>
```

And specify the integer value for `sos_drawing_width` (alongside your other resources):

```
<integer name="sos_drawing_width">10</integer>
```

Now the application annotation color looks like this:



Name Jane Ballard

S.I.N 111-222-333

Since 2014-05-11

SOS: Toast Behavior

The SOS session provides context to the customer through toasts for various events over the lifetime of the session. You can customize these toasts.

There are two ways to customize the toast behavior.

Change the Toast Text

You can change any toast text by overriding the string resource that the toast uses. See Customize and Localize Strings with the Service SDK for more information about overriding string resources.

Disable Some or All Toasts

If you do not want a particular toast to ever appear during a session, you can disable the toast with `disableToasts`. This method takes an `EnumSet` of `SosToast` elements. Some built-in `enum` sets are already defined in the `SosToast` Javadoc.

For example, this code disables all toasts over the course of the session:

```
SosOptions options = new SosOptions(
  'pod',          // replaced with your real pod identifier
  'orgId',        // replaced with your real orgId
  'deploymentId' // replaced with your real deploymentId
);

SosConfiguration config = SosConfiguration.builder()
    .disableToasts(EnumSet.allOf(SosToast.class))
    .build();

Sos.session(opts)
    .configuration(config)
    .start(this);
```

Troubleshooting the Service SDK

Get some guidance when you run into issues.

Enable Debug Logging for the Android SDK
SDK logs are disabled by default. To enable logging, you add a sink and then specify a log level.

Can't Access My Knowledge Base
What to do when you can't get to your knowledge base from within your app.

Can't Connect to Chat
If you can't make a successful connection from your app, even when an agent is standing by, review how you've set up your chat implementation.

Error Using the Salesforce Mobile SDK with the Service SDK
If you're trying to build a Service SDK project that explicitly embeds the Salesforce Mobile SDK, exclude these two maven dependencies to prevent conflicts.

SOS Network Troubleshooting Guide
If you can't connect with an SOS agent from your app, you have network connectivity issues, possibly related to your firewall or proxy.

Enable Debug Logging for the Android SDK

SDK logs are disabled by default. To enable logging, you add a sink and then specify a log level.

Call `addSink` to direct the debug logs to the specified sink. `LOG_CAT_SINK` directs all log messages to the Android logcat using the standard `Log.<level>` calls. Set the log level with `setLogLevel`. For example:

```
ServiceLogging.addSink(ServiceLogging.LOG_CAT_SINK);
ServiceLogging.setLogLevel(ServiceLogging.LEVEL_TRACE);
```

To direct the logs somewhere other than logcat, implement your own `ServiceLoggingSink`.

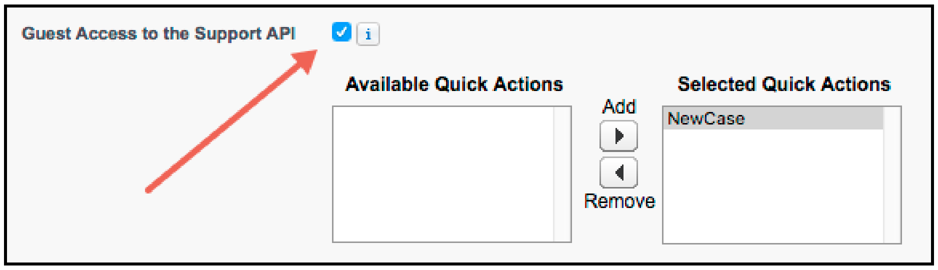
To learn more about Android logging, see Write and View Logs.

Can't Access My Knowledge Base

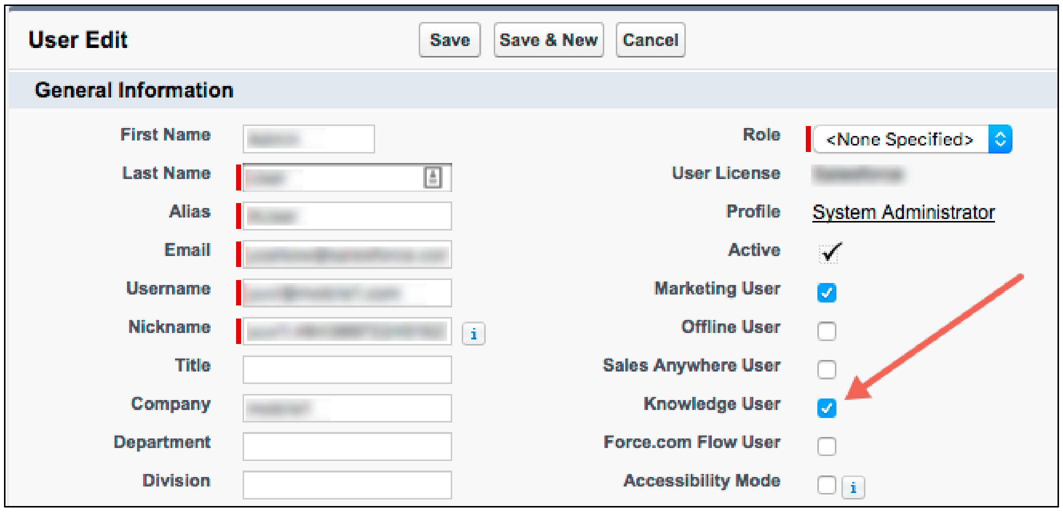
What to do when you can't get to your knowledge base from within your app.

Run through this checklist to help diagnose the root cause.

1. Have you set up a Experience Cloud or Salesforce site? See Cloud Setup for Knowledge for more info.
2. Do you have **Guest Access to the Support API** enabled for your site? See Cloud Setup for Knowledge for more info.


Guest Access to the Support API ✓
Available Quick Actions
Selected Quick Actions: NewCase

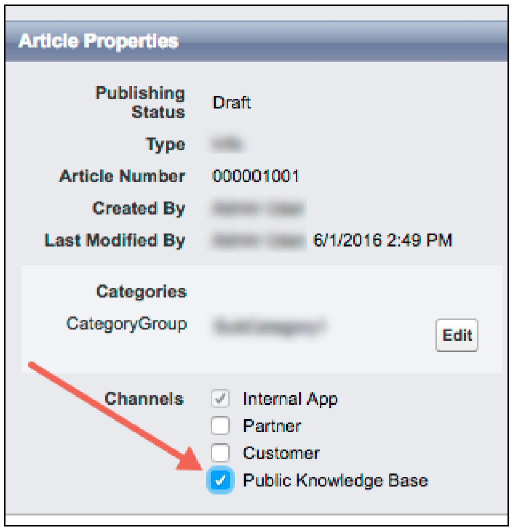
3. (For Knowledge only) Do you have **Knowledge** enabled in your org? Do you have Knowledge licenses? See Cloud Setup for Knowledge for more info.
4. (For Knowledge only) Is the user setting up the knowledge base enabled as a **Knowledge User**? See Cloud Setup for Knowledge for more info.


User Edit

General Information

First Name		Role	<None Specified>
Last Name		User License	
Alias		Profile	System Administrator
Email		Active	✓
Username		Marketing User	✓
Nickname		Offline User	
Title		Sales Anywhere User	
Company		Knowledge User	✓
Department		Force.com Flow User	
Division		Accessibility Mode	

5. (For Knowledge only) Have you made the article types, the data categories, and the article layout fields visible to guest users? See Guest User Access for Your Experience Cloud Site for more info.
6. (For Knowledge only) Have you made your articles accessible to the **Public Knowledge Base** channel? See Cloud Setup for Knowledge for more info.


Article Properties

Publishing Status	Draft
Type	
Article Number	000001001
Created By	
Last Modified By	6/1/2016 2:49 PM

Categories

CategoryGroup

Channels: [✓ Internal App] Partner Customer [✓ Public Knowledge Base]

Can't Connect to Chat

If you can't make a successful connection from your app, even when an agent is standing by, review how you've set up your chat implementation.

Run through this checklist to help diagnose the root cause.

1. If you're using the default UI for chat, verify that you are calling `showChat` on the main UI thread.
2. Verify that the chat endpoint in your code only specifies the hostname. For instance, if your endpoint is `https://`***`MyDomainName`***`.my.salesforcescrt.com/chat/rest/`, then use the following value in your code: ***`MyDomainName`***`.my.salesforcescrt.com`.

Note: If you don't have a My Domain deployed in your org, your URL formats are different. For details, see My Domain URL Formats in Salesforce Help.

3. Verify that you're using the correct chat endpoint. See Get Chat Settings from Your Org for more info.
4. Verify that you're using the correct deployment ID and button ID. See Get Chat Settings from Your Org for more info.
5. Verify that you've correctly set up your chat implementation. See Org Setup for Chat in Lightning Experience with a Guided Flow for more info.

Error Using the Salesforce Mobile SDK with the Service SDK

If you're trying to build a Service SDK project that explicitly embeds the Salesforce Mobile SDK, exclude these two maven dependencies to prevent conflicts.

When building an app with both SDKs, you may encounter an error such as this error:

```
Error: more than one library with package name 'com.salesforce.androidsdk.analytics'
```

Or this error:

```
Duplicate zip entry
[classes.jar:com/salesforce/androidsdk/smartstore/app/SmartStoreSDKManager.class]
```

To solve the problem, exclude the problematic dependencies from your `build.gradle` file.

```
compile('com.salesforce.service:servicesdk:224.1.0')
{
    exclude group: 'com.salesforce.mobilesdk', module: 'SmartStore'
    exclude group: 'com.salesforce.mobilesdk', module: 'SalesforceSDK'
}
```

SOS Network Troubleshooting Guide

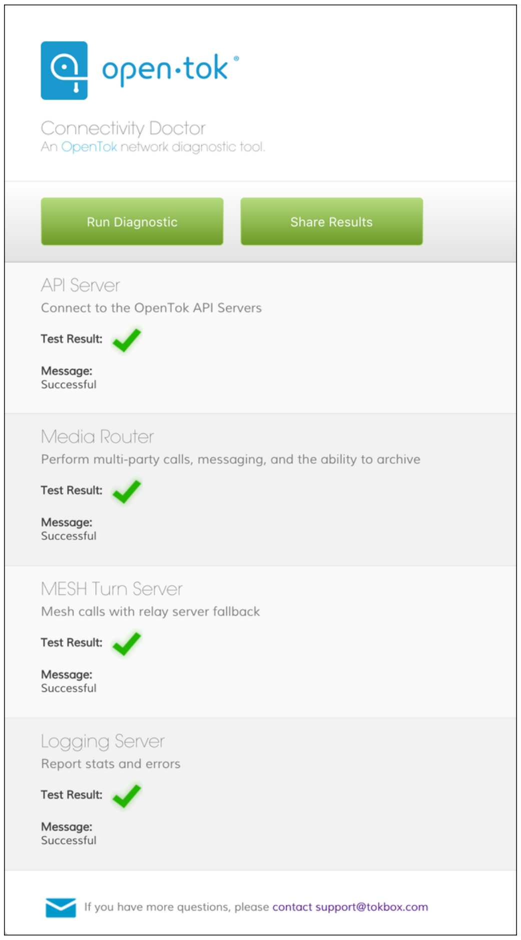
If you can't connect with an SOS agent from your app, you have network connectivity issues, possibly related to your firewall or proxy.

SOS uses the Tokbox OpenTok platform to provide screen sharing and video communication during an SOS session. These guidelines can help you diagnose whether the problem is linked to a networking issue and how to send us diagnostic information if necessary.

Step 1: Run Connectivity Doctor

The Tokbox Connectivity Doctor tests for connectivity issues. You can access this tool via the web, an iOS app, or an Android app. This tool tests network issues in these areas.

1. API server – Session initialization and signaling tests
2. Media router – Whether you can access Tokbox media servers
3. MESH turn server – Relay server fallback mechanism
4. Logging server – Communication of stats and errors to the Tokbox logging server



Connectivity Doctor
An OpenTok network diagnostic tool

API Server
Connect to the OpenTok API Servers
Test Result: ✓
Message:
Successful

Media Router
Perform multi-party calls, messaging, and the ability to archive
Test Result: ✓
Message:
Successful

MESH Turn Server
Mesh calls with relay server fallback
Test Result: ✓
Message:
Successful

Logging Server
Report stats and errors
Test Result: ✓
Message:
Successful

If you have more questions, please contact support@tokbox.com

If all tests pass, go to step 2. If any test fails, you probably need to configure your ports.

API Server or Logging Server Issues

OpenTok clients use HTTP and WSS connections from the client browser to the OpenTok servers on port **TCP/443**. If the only way to access the internet from your network is through a proxy, it must be a transparent proxy. Make sure that TCP/443 is open.

Media Router or Mesh Turn Issues

OpenTok clients can use UDP or TCP connections for media. Salesforce recommends that UDP is enabled to improve the quality of real-time audio and video communications. This connection is bidirectional but always initiated from the client so an external entity can't send malicious traffic in the opposite direction.

- Best experience: We recommend that you open **UDP ports 1025 - 65535**.
- Good experience: Open **UDP port 3478**.
- Minimum experience: Open **TCP port 443**. Some firewall or proxy rules only allow for SSL traffic over port 443. Make sure that non-web traffic can also pass over this port.

Step 2: Test the Tokbox Chat Room

Tokbox has a public-facing chat room site (https://opentokrtc.com/) that you can use for normal chatting. You can also use this site as a test tool.

1. From the chat room site, create a room called "example". Join that room or click here. You should then see yourself on video. If the video isn't present, make sure that you've given access to the camera and microphone. If nothing happens, go to step 3.

2. Open another browser tab and enter the same URL link as in the previous browser tab (for example, https://opentokrtc.com/room/example). You should see a two-way audio and video-enabled chat. If this process doesn't work, go to step 3.

If you can have a two-way chat, your ports are configured properly and you're done.

Step 3: Gather JSON Metadata

In this step, we'll gather metadata about the failed chat room session. To get this information, open a browser tab and enter https://opentokrtc.com/example.json. If you created a room with a different name, replace the word "example" in the URL accordingly. You'll see JSON content similar to this example.

```
{"apiKey":"45599822","token":"T1==cGyydG5lcl9pZD00NTU5OTgyMiZzaWc9NTcyOWI4NjJhOT
diN2EwYWRmMjZkZjI5MzkxZjkwMjdlNmM0ODNiMTpzZXNzaW9uX2lkPTFfTVg0ME5UVTVPVGd5TW41LU
1UUTNPVEUyTWpVek1USTVObjQ0WkcxSFp6VnNVMnBMZFRKVFp6SmhaRlZ6WVhvM1dVTi1mZyZjcmVhdG
VfdGltZT0xNDc5MTY0MzA4Jm5vbmNlPTAuMzgzODc1MjEwMzAzODEzMiZyb2xlPXB1Ymxpc2hlciZleH
BpcmVfdGltZT0xNDc5MjUwNzA4JmNvbm5lY3Rpb25fZGF0YT0lN0IlMjJ1c2VyTmFtZSUyMiUzQSUyMk
Fub255bW91cyUyMFVzZXI3MiUyMiU3RA==","username":"Anonymous User72","firebaseURL":
"https://ot-archiving.firebaseio.com/sessions//1_MX40NTU5OTgyMn5-MTQ3OTE2MjUzMTI
5Nn43ZG1HZzVsU2pLdTJTZzJhZFVzYXo3WUN-fg","firebaseToken":"eyJ0eXAiOiJKV1QiLCJhbG
ciOiJIUzI1NiJ9.eyJ2IjowLCJkIjp7InVpZCI6IkFub255bW91cyBVc2VyNzIwLjc3NDEwOTM5OTg3N
zQ4ODYiLCJzZXNzaW9uSWQiOiIxX01YNDBOVFU1T1RneU1uNS1NVFEzT1RFMk1qVXpNVEk1Tm40NFpHM
UhaelZzVTJwTGRUSlRaekpoWkZWellYbzNXVU4tZmciLCJyb2xlIjoidXNlciIsIm5hbWUiOiJBbm9ue
W1vdXMgVXNlcjcyIn0sImlhdCI6MTQ3OTE2NDMwN30.ep6l4x_3VGegXVTfwpaOYMGRGOYI944w5Og1h
TyDPfQ","chromeExtId":"undefined","sid":"1_M440NTU5OTgyMn5-MTQ3OTE2MjUzMTI5Nn44Z
G1HZzVsU2pLdTJTZzJhZFVzYXo3WUN-fg"}
```

The output has key/value pairs for the API key ("apiKey"), token ("token"), and session ID ("sid"). Save this information. Tokbox monitors the entire flow for failed and successful sessions (only if you were able to successfully verify the Connectivity Doctor ports requirements first). We'll use this metadata to debug your issue. Go to step 4.

Step 4: Open Support Ticket

Please create a Salesforce support ticket or contact your account team for more help.

Answer the following questions in your support request.

1. TOPOLOGY: What is your network topology? How does data flow through the topology to reach our cloud?
2. ENVIRONMENT: Are you using a virtual environment?
3. PROXY & FIREWALL: What is the type and name of your proxy? What are your firewall restrictions? If you are using a proxy can you execute step 1 and step 2 just through a firewall?
4. PLATFORM & VERSION: What platform are you using? What is the OS version? Which SDK are you using? Which SDK version?
5. DESCRIPTION: Describe the problem you encountered, and the steps you took to try to resolve the problem.
6. LOGS: Can you provide us with any error logs from your side or any other information you deem fit for the problem? Include any relevant network traces or screenshots.
7. CONNECTIVITY DOCTOR: What Connectivity Doctor tests failed? Did you open the required ports and still have issues?
8. JSON METADATA: If applicable, send us the Key/Token/Session information captured in step 3.

Data Protection and Security in the Service SDK for Android

The Service SDK does not collect or store personal data from its users. We ensure that data is secure both locally and when in transit.

- **Secure data at rest**. We don't store personal data about the user. We manage keys using the Android Keystore. All content fetched from Salesforce servers is stored locally using AES-256 encryption. You can explicitly delete knowledge data by following these instructions: Cache and Encryption Behavior for Knowledge. You can explicitly delete case management data by following these instructions: Cache and Encryption Behavior for Case Management.
- **Secure data in transit**. All network communication occurs over SSL using TLS 1.2.

Reference Documentation

Reference documentation for Service SDK for Android.

To access the reference documentation for the Service SDK for Android, visit:

- forcedotcom.github.io/ServiceSDK-Android

This site contains API documentation for the latest version of the SDK.

Reference Index
A list of all classes, interfaces, methods, constants, and enums referenced from this developer's guide.

Reference Index

A list of all classes, interfaces, methods, constants, and enums referenced from this developer's guide.

Chat Index

- `AgentAvailabilityClient`
- `AgentListener`
- `AvailabilityState`
- `ChatAnalytics`
- `ChatClient`
  - `addFileTransferRequestListener`
- `ChatConfiguration`
  - `ChatConfiguration.Builder`
    - `chatEntities`
    - `chatUserData`
    - `maximumWaitTime`
    - `minimumWaitTime`
    - `queueStyle`
- `ChatCore`
- `ChatEndReason`

- `ChatEntity`
- `ChatEntityField`
- `ChatEventListener`
- `ChatKnowledgeArticlePreviewClickListener`
- `ChatKnowledgeArticlePreviewDataHelper`
- `ChatKnowledgeArticlePreviewDataProvider`
- `ChatSessionState`
- `ChatUI`
- `ChatUIClient`
- `ChatUIConfiguration`
  - `ChatUIConfiguration.Builder`
    - `chatBotAvatar`
    - `enableChatBotBanner`
- `ChatUserData`
- `FileTransferAssistant`
- `FileTransferRequestListener`
- `FileTransferStatus`
- `PreChatPickListField`
- `PreChatTextInputField`
- `QueueListener`
- `SessionInfoListener`
- `SessionStateListener`

Knowledge Index

- `ArticleSummary`
  - `getArticleId`
  - `getArticleNumber`
- `DataCategorySummary`
  - `getName`
- `KnowledgeClient`
- `KnowledgeConfiguration`
  - `KnowledgeConfiguration.Builder`
    - `offlineResourceConfig`
- `KnowledgeCore`
  - `deleteCache`
- `KnowledgeCssProvider`
- `KnowledgeImageProvider`

Case Management Index

SOS Index

- `View`

Common Index

- `Async`
- `AuthenticatedUser`
- `AuthTokenProvider`
- `PushNotificationListener`
  - `onPushNotificationReceived`
- `SalesforceConnectedApp`
- `ServiceAnalytics`
- `ServiceAnalyticsListener`
- `ServiceLogging`
  - `addSink`
  - `setLogLevel`
- `ServiceLoggingSink`

Resource Files

- Chat String Resources
- Knowledge String Resources
- Case Management String Resources
- SOS String Resources
- Common String Resources

Additional Resources

If you're looking for other resources, check out this list of links to related documentation.

- Embedded Service SDK for Mobile Apps Resources
  - Service SDK Developer Center
  - Service SDK Trailhead Learning Module
  - iOS: Release Notes, Dev Guide, Reference Docs, Examples
  - Android: Release Notes, Dev Guide, Reference Docs, Examples
- General Resources
  - Service Cloud Developer Center
  - Salesforce Developer Documentation
  - Salesforce Help

INDEX