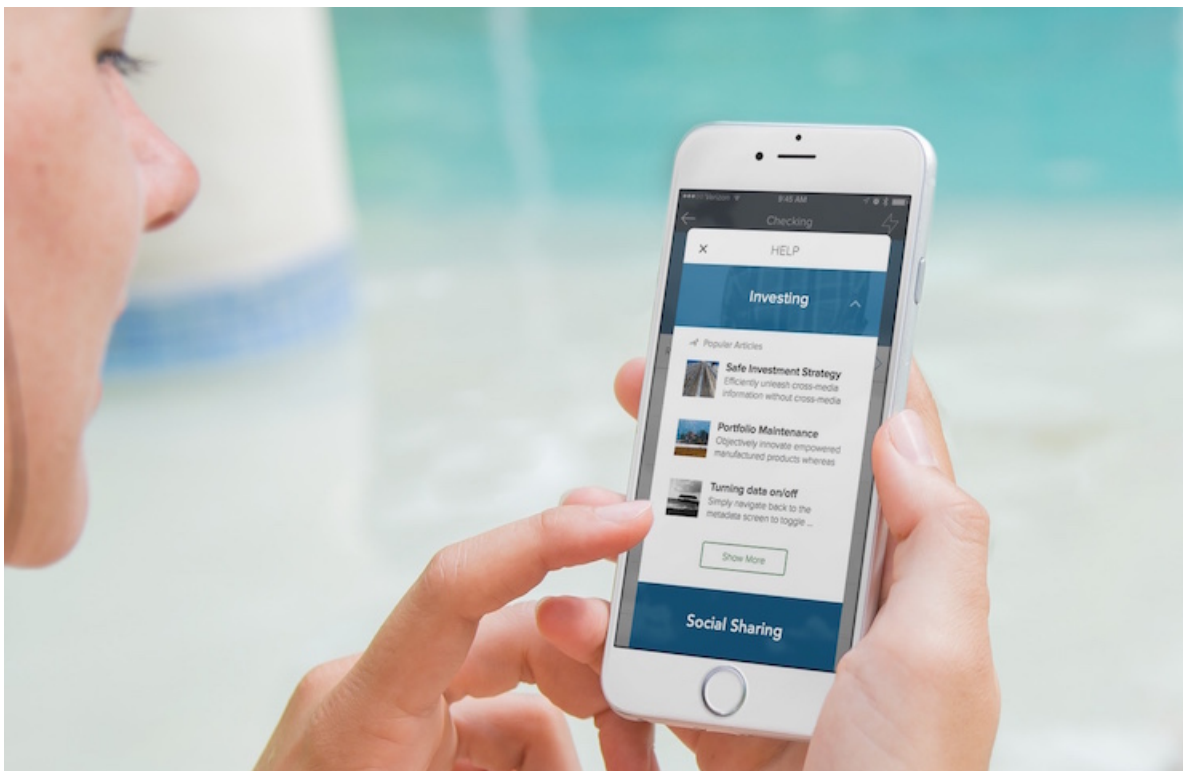




Embedded Service Chat SDK for Android Developer Guide

Version 224.2



CONTENTS

- [Embedded Service Chat SDK for Android Developer Guide](#) 1
- Release Notes 2
- Service Cloud Setup 2
- SDK Setup 12
- Android Examples 25
 - Chat 25
- SDK Customizations 56
- Troubleshooting 62
- Data Protection and Security 64
- Reference Documentation 64
- Additional Resources 69
- [Index](#) 70

EMBEDDED SERVICE CHAT SDK FOR ANDROID DEVELOPER GUIDE

The Embedded Service Chat SDK for Mobile Apps makes it easy to give customers access to powerful chat features right from within your native app. This guide helps you get started using the SDK in your mobile app.



Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

October 2023 Release (Version 224.2.6)

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

Component	Version Number
Chat	4.3.6
Common	8.0.6

[Release Notes](#)

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

[Service Cloud Setup for the Embedded Service Chat SDK for Mobile Apps](#)

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

[Embedded Service Chat 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 Chat SDK.

[Using Chat with the Service Chat SDK](#)

Add the Chat experience to your mobile app.

[SDK Customizations with the Service Chat SDK for Android](#)

Once you've played around with some of the SDK features, use this section to learn how to customize the Service Chat 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 Chat SDK](#)

Get some guidance when you run into issues.

[Data Protection and Security in the Service Chat SDK for Android](#)

The Service Chat 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 Chat SDK for Android.

[Additional Resources](#)

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

SEE ALSO:

[Service SDK for Android Release Notes](#)

[Service SDK for Android Reference Documentation](#)

[Service SDK for iOS Developer Guide](#)

Release Notes

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



Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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

Service Cloud Setup for the Embedded Service Chat SDK for Mobile Apps

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



Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

[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.

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.

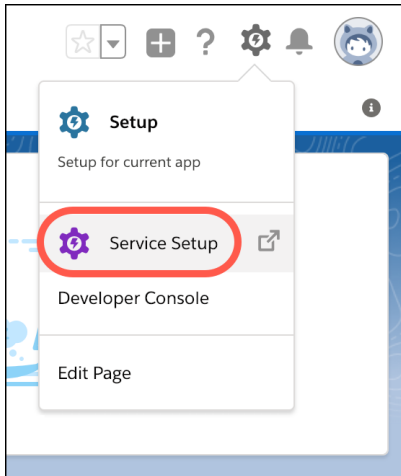


Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

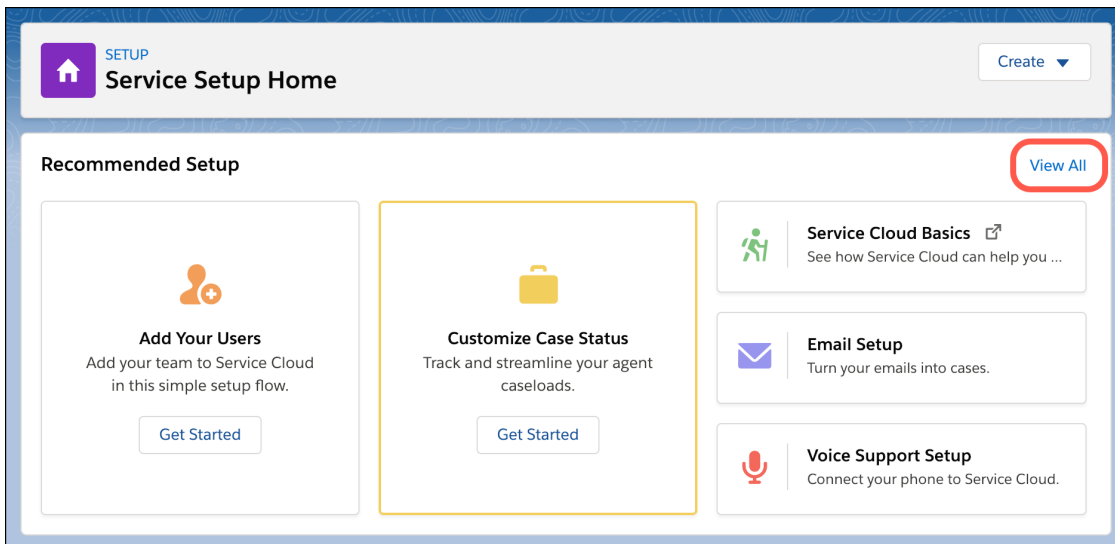
 **Warning:** If you use the [Permitted Domains setting in your Chat deployment](#), you'll get unreliable information from the chat availability check in the SDK. For instance, the agent availability status may always return `false`. If you want to use Permitted Domains for your web chat deployment, we strongly advise that you create a separate deployment for the Service SDK.

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**.



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

Back

Next

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

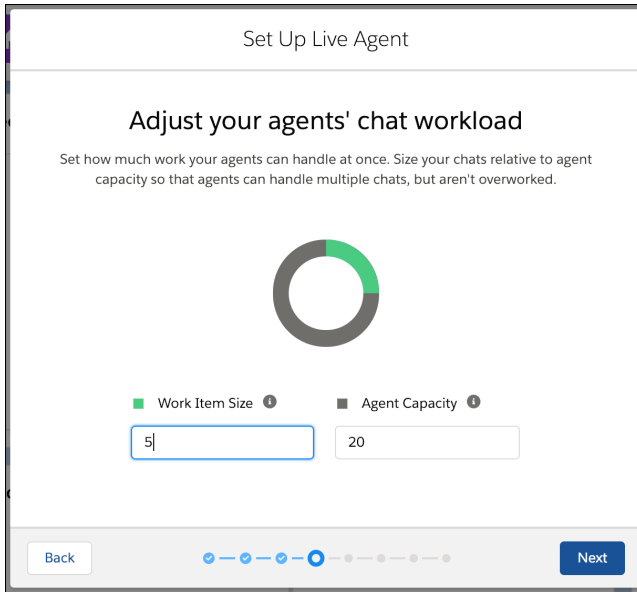
Name

Priority

Back

Next

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

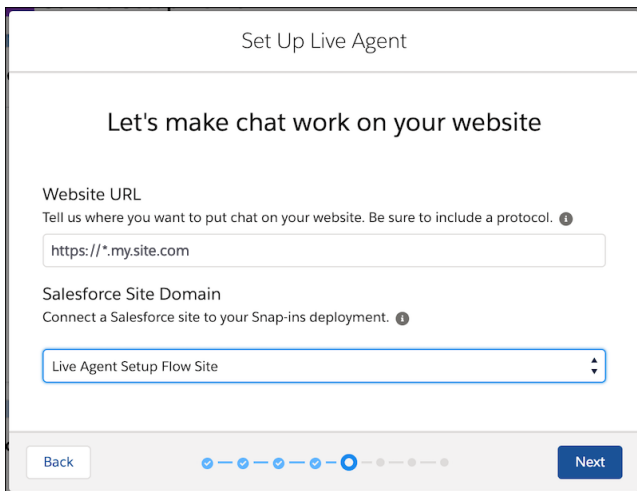


The screenshot shows the 'Set Up Live Agent' screen with the heading 'Adjust your agents' chat workload'. Below the heading is a sub-header: 'Set how much work your agents can handle at once. Size your chats relative to agent capacity so that agents can handle multiple chats, but aren't overworked.' A donut chart is displayed, with a green segment representing 'Work Item Size' and a grey segment representing 'Agent Capacity'. Below the chart are two input fields: 'Work Item Size' with the value '5' and 'Agent Capacity' with the value '20'. At the bottom, there are 'Back' and 'Next' buttons, and a progress indicator showing the current step is selected.

8. For the website URL, enter either:

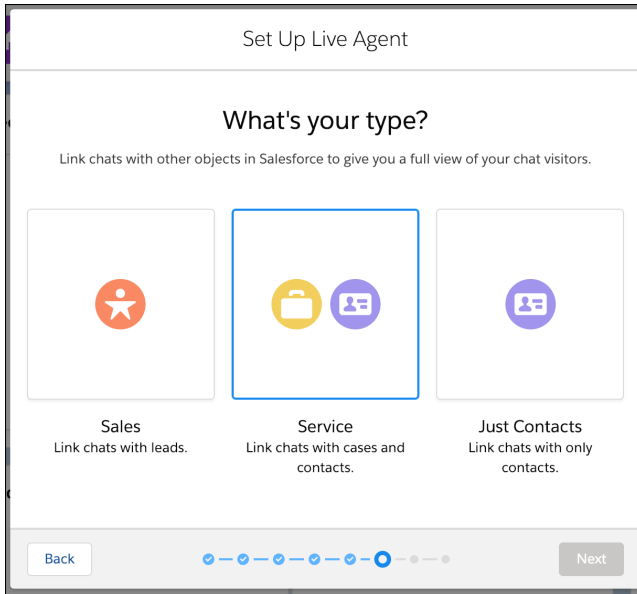
- a. The URL of your site.
- b. `https://`, followed by the last part of your site's URL: `https://*.my.site.com`, `https://*.salesforce-sites.com`, or `https://*.force.com`.

Then create or select a site.

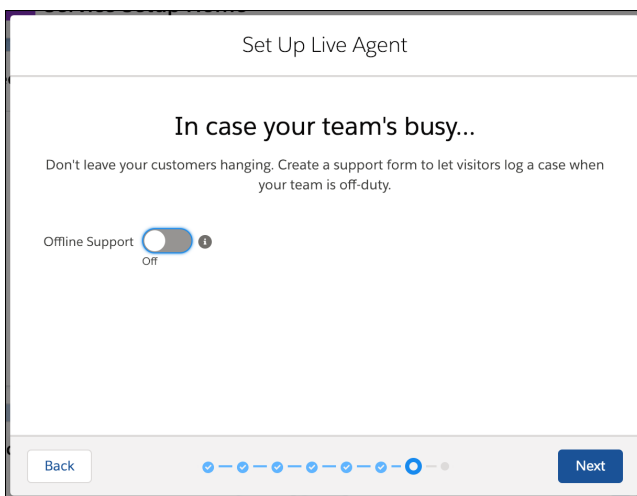


The screenshot shows the 'Set Up Live Agent' screen with the heading 'Let's make chat work on your website'. Below the heading are two sections: 'Website URL' with the instruction 'Tell us where you want to put chat on your website. Be sure to include a protocol.' and a text input field containing 'https://*.my.site.com'; and 'Salesforce Site Domain' with the instruction 'Connect a Salesforce site to your Snap-ins deployment.' and a dropdown menu showing 'Live Agent Setup Flow Site'. At the bottom, there are 'Back' and 'Next' buttons, and a progress indicator showing the current step is selected.

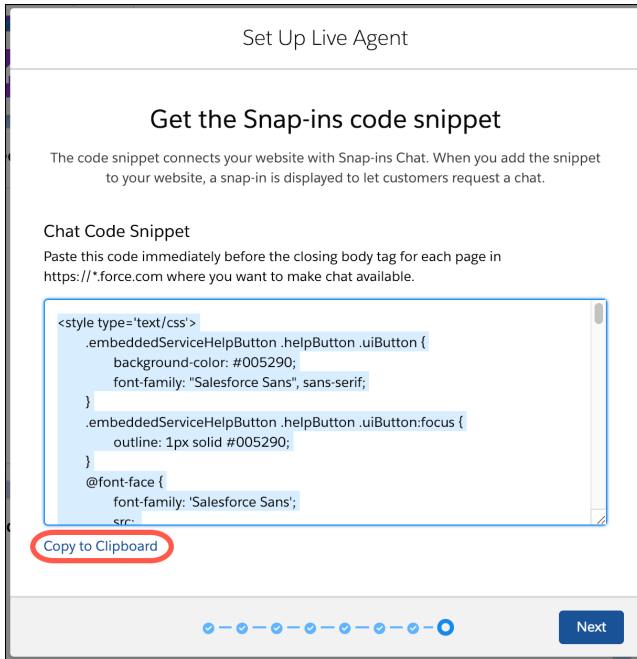
9. For the type of chat, select **Service**.



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



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.




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

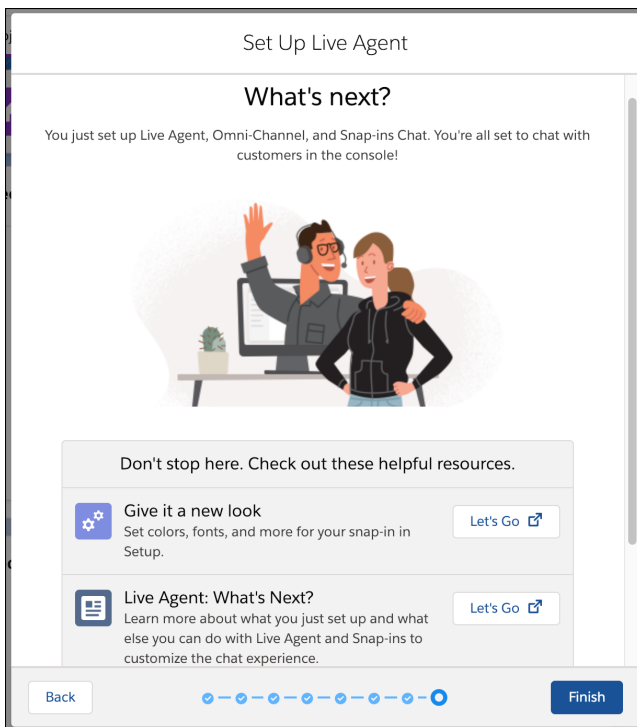
- (1) Chat Endpoint Hostname—This value is the hostname of the `baseLiveAgentURL` property. When copying the hostname, be sure not to include the protocol or the path. For instance, if the value for `baseLiveAgentURL` is `https://MyDomainName.my.salesforce-scr.t.com/chat`, then the hostname is **MyDomainName**.my.salesforce-scr.t.com.
- (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', 2,  
    'Chat_Agents',  
    {  
        baseLiveAgentContentURL: 'https://c.ph2.salesforceliveagent.com/content',  
        deploymentId: '5722U000000Dt72', 3,  
        buttonId: '5731U000000E32v', 4,  
        baseLiveAgentURL: 'https://d.la2.salesforceliveagent.com/chat',  
        eswLiveAgentDevName: 'Chat_Agents', 1,  
        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**.



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

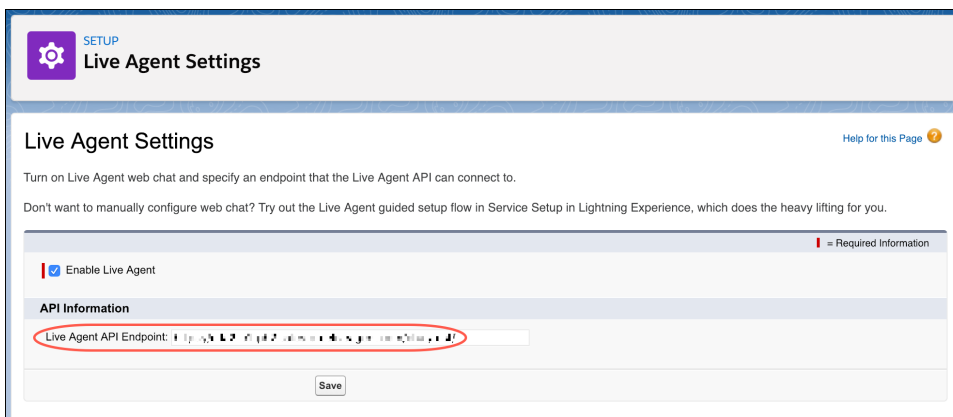
After you've set up chat in the console, supply your app developer with four values: the chat endpoint hostname, the organization ID, the deployment ID, and the button ID. You can get this information from your org's setup.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

 **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**.



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**.

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 [Edit] [Deactivate Org]

Organization Name	Salesforce	Phone	
Primary Contact		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	<input type="checkbox"/>	Currency Locale	English (United States) - USD
Newsletter	<input checked="" type="checkbox"/>	Used Data Space	436 KB (9%) [View]
Admin Newsletter	<input checked="" type="checkbox"/>	Used File Space	13 KB (0%) [View]
Hide Notices About System Maintenance	<input type="checkbox"/>	API Requests, Last 24 Hours	1 (15,000 max)
Hide Notices About System Downtime	<input type="checkbox"/>	Streaming API Events, Last 24 Hours	0 (10,000 max)
		Restricted Logins, Current Month	0 (0 max)
		Salesforce.com Organization ID	573B000000005KXz
		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.

Deployments

Live Agent Deployments

Agent Configuration [Edit]

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='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', '573B000000005KXz',
'00DB000000003Rxz');
</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', '573B000000005KXz',
'00DB000000003Rxz');
</script>
```

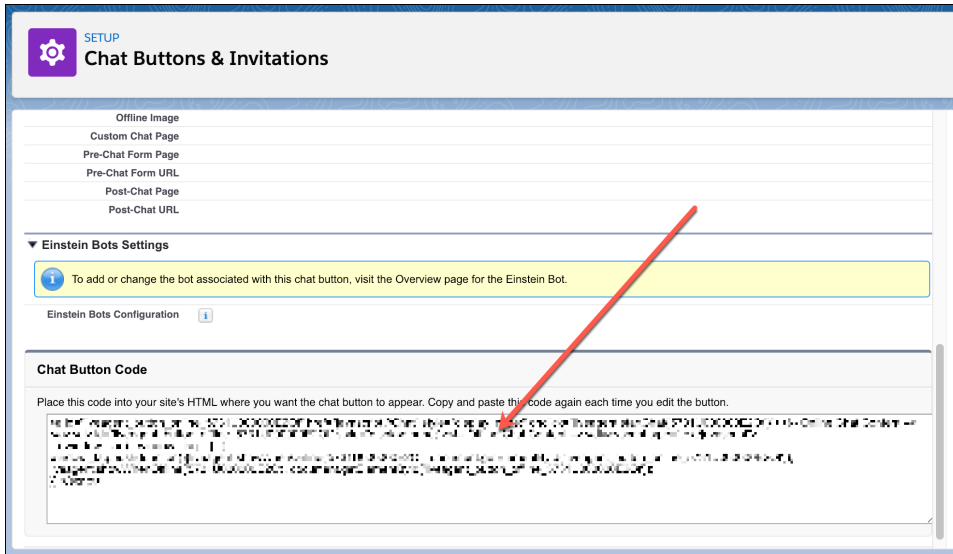
The deployment ID value is:

573B000000005KXz

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.



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.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid

service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

 **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](#).

Embedded Service Chat SDK for Mobile Apps Setup

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

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

[Requirements for the Service Chat SDK for Android](#)

The Salesforce org, SDK development, and mobile app requirements for using the Service Chat SDK.

[Accessibility with the Service Chat SDK for Android](#)

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

[Data Collection for the Service Chat SDK](#)

The Service Chat SDK collects and transmits data to perform basic operations. This data falls into three categories: pre-chat data, chat message data, and logging data.

[Install the Service SDK for Android](#)

Install the Service SDK for Android using Gradle.

[Authentication with the Service Chat SDK for Android](#)

The Service Chat 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 Chat 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 Chat SDK using your `PushNotificationListener` implementation.

[Analytics with the Service Chat SDK for Android](#)


You can listen to user-driven events from the Service Chat 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 Chat SDK for Android

The Salesforce org, SDK development, and mobile app requirements for using the Service Chat SDK.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

Salesforce Org Requirements

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

The SDK inserts permission requirements into the manifest of your compiled application package.

Permission
General Purpose Permissions
<code>android.permission.ACCESS_NETWORK_STATE</code>
<code>android.permission.INTERNET</code>
<code>android.permission.READ_EXTERNAL_STORAGE</code>
<code>android.permission.WRITE_EXTERNAL_STORAGE</code>
Real-Time Chat and Video Permissions
<code>android.permission.CAMERA</code>

Accessibility with the Service Chat SDK for Android

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

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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 Chat 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 Chat SDK](#).

Known Issues

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

Data Collection for the Service Chat SDK

The Service Chat SDK collects and transmits data to perform basic operations. This data falls into three categories: pre-chat data, chat message data, and logging data.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

Pre-Chat Data

Data types collected:

- Personal info (for example, name, email). Most pre-chat forms contain some personal info.
- Other info. Other data collected depends on how you design your pre-chat form.

All data specified in the pre-chat form is securely transmitted to your Salesforce org using HTTPS communication (TLS 1.3). This data is stored in your org indefinitely, depending on how you handle pre-chat information in your org. You can change this behavior from within your org. You can even remove the pre-chat form altogether.

Chat Message Data

Data types collected:

- In-app messages. User chat messages get sent to agents through your Salesforce org.
- Photos. Users can send photos to agents.
- Other info. Any content a user types into a message gets sent to an agent or chatbot.

All user messages are securely transmitted to your Salesforce org using HTTPS communication (TLS 1.3). This data is stored in your org indefinitely. You can remove any of this data from your org.

Logging Data

Data types collected:


- App interactions
- Other actions

For logging purposes, we send anonymized information to Splunk servers using HTTPS communication (TLS 1.3). This data doesn't contain any customer-identifiable or hardware-identifiable information. We log info about how users interact with the SDK and we log some basic system information (such as battery stats) while they use the SDK.

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 Chat SDK for Android](#).

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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	<code>com.salesforce.service:servicesdk:224.2.6</code>	https://sfdc-maven.salesforce.com/android/maven2/com/salesforce/service/servicesdk/
Chat	<code>com.salesforce.service:chat-ui:4.3.6</code> (if you're only using Chat Core, then use <code>com.salesforce.service:chat-core:4.3.6</code>)	https://sfdc-maven.salesforce.com/android/maven2/com/salesforce/service/chat-ui/

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

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

 **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:224.2.6'
}
```

2. Declare permissions.


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"/>
```

You can now start using the Service SDK for Android.

Authentication with the Service Chat SDK for Android

The Service Chat 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.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

AuthenticatedUser

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

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

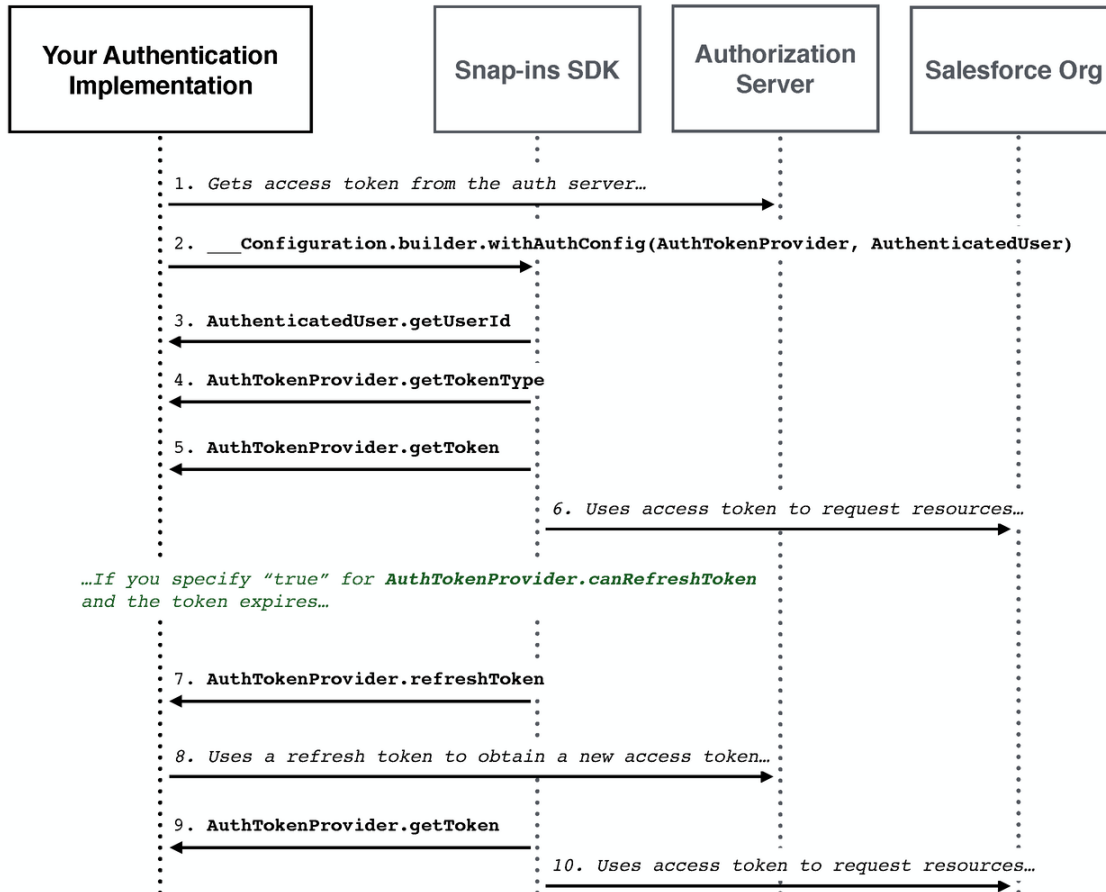
AuthTokenProvider

`AuthTokenProvider` is an interface to the system that obtains the access token. If the access token expires, the Service Chat 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.



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 Chat 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 Chat SDK using the provided authentication mechanism within the Salesforce Mobile SDK.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid

service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

Before starting, make sure that you've already:

- Installed the Service Chat 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 Chat 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 Chat 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 Chat SDK as an authenticated user.

Push Notifications with the Service Chat 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 Chat SDK using your `PushNotificationListener` implementation.


 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

The following activities can cause notifications:

- Agent has connected
- Agent sent a message
- Agent requested a file transfer
- Agent canceled a file transfer
- Agent ended a session

To implement notifications, follow these steps:

1. Set up authentication in your app. To learn more, see [Authentication with the Service Chat 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 Chat SDK `connected-app` dependency in your module's `build.gradle` file.

```
dependencies {  
  
    // Add the connected-app dependency  
    implementation "com.salesforce.service:connected-app:8.0.6"  
  
    // ... 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 Chat 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](#).
5. Implement `PushNotificationListener` and handle the push notification event in the `onPushNotificationReceived` method.
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 Chat SDK for Android

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



Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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.

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

# ----- OkHttpClient -----
-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>;
}
```

Android Examples

Use these examples to learn more about the Service Chat SDK.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

Check out our [sample apps on GitHub \(github.com/forcedotcom/ServiceSDK-Android\)](https://github.com/forcedotcom/ServiceSDK-Android).

Using Chat with the Service Chat SDK

Add the Chat experience to your mobile app.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

[Chat in the Service Chat SDK for Android](#)

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

[Quick Setup: Chat in the Service Chat 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.

[Handle Custom URLs in Chat](#)

Have your agents pass along custom URLs to perform specific actions in your mobile app.

[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 Chat SDK for Android

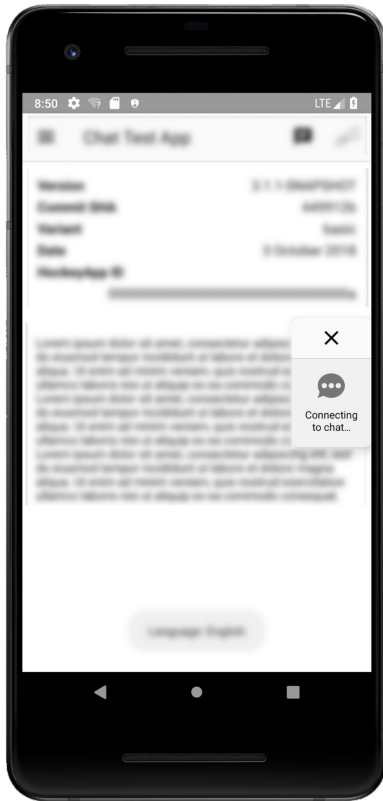
Using Chat within the Service Chat SDK, you can provide real-time chat sessions from within your native app.



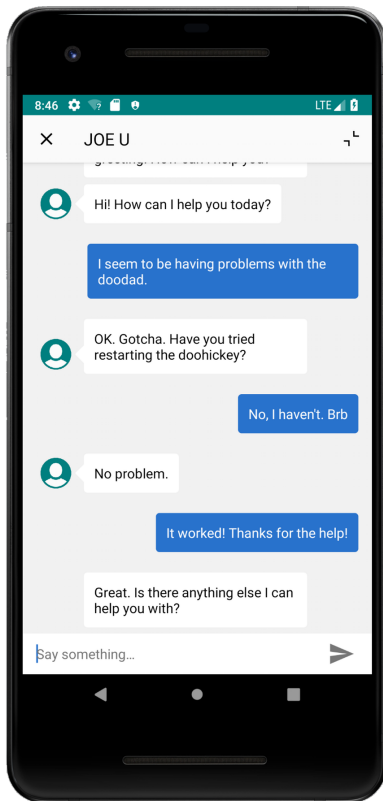
Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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.



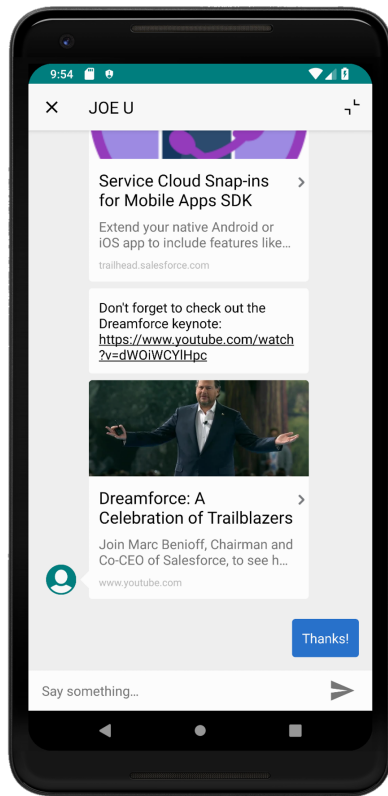
When the user taps the chat thumbnail, the app goes full screen.



If you've implemented Einstein Bots, the user first [speaks with a chat bot](#) on page 32 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.



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 Chat SDK

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

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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 59 (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.

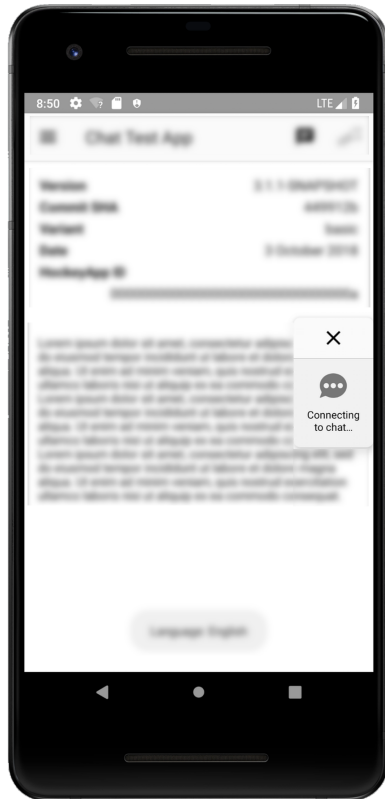
5. (Optional) Add listeners for state changes or events.

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](#).

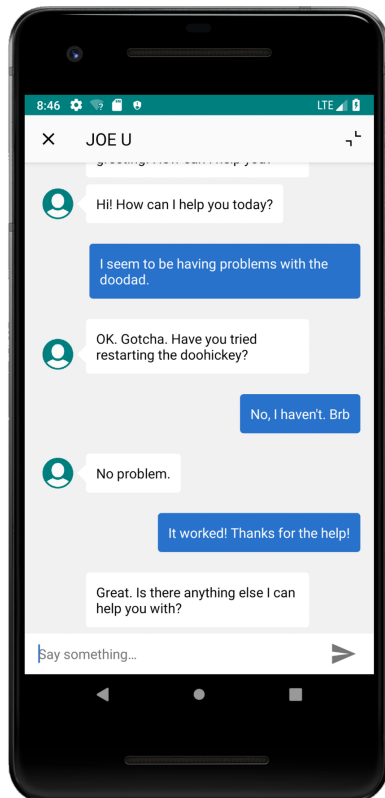



Warning: If you use an incorrect button ID in your `ChatConfiguration`, the chat fails and your `SessionStateListener` reports a `ChatEndReason` of `NetworkError`.

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.



 **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.

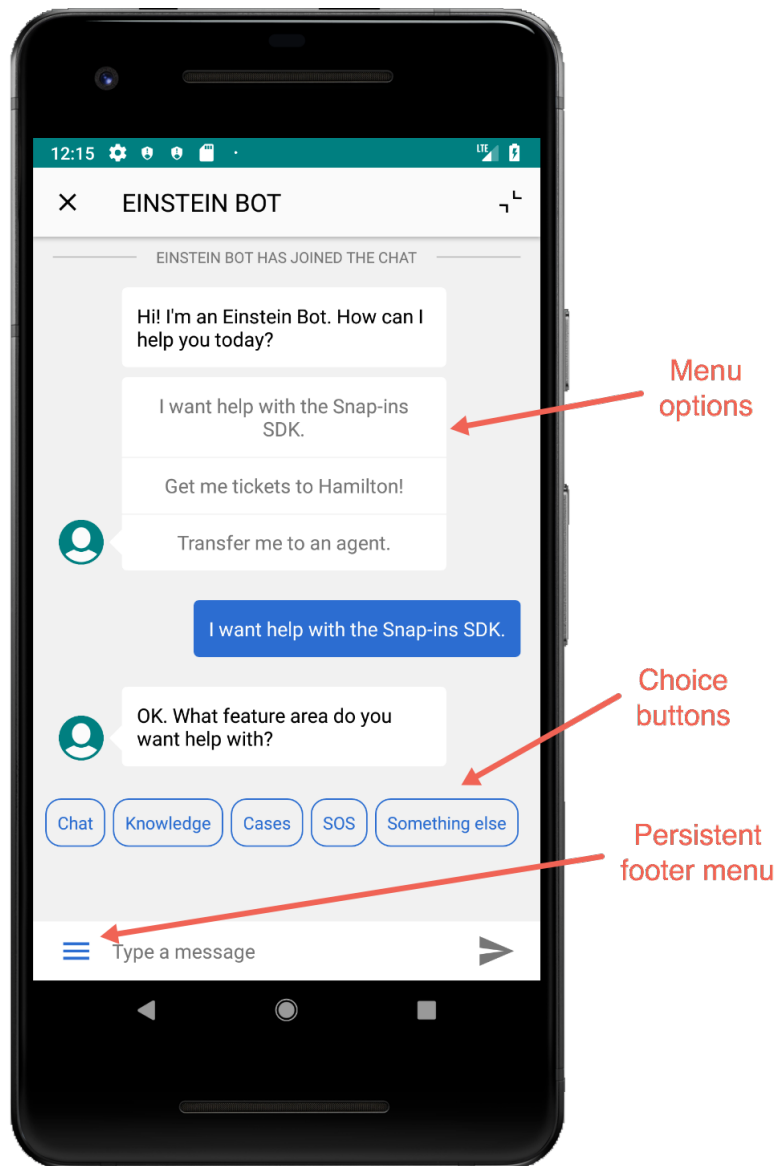
Use Einstein Bots with Chat

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.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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.




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

Feature Area	Details
Einstein Bot Avatar	<p>Configure the bot avatar that displays during a session with a bot. To do this, add your <code>Drawable</code> to the <code>ChatUIConfiguration</code> builder using the <code>chatBotAvatar</code> method.</p> <pre>final ChatUIConfiguration.Builder uiConfigBuilder = new ChatUIConfiguration.Builder(); uiConfigBuilder.chatBotAvatar(R.drawable.my_chatbot_avatar); uiConfigBuilder.build();</pre>

Feature Area	Details
Einstein Bot Banner	<p>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.</p> <pre>final ChatUIConfiguration.Builder uiConfigBuilder = new ChatUIConfiguration.Builder(); uiConfigBuilder.enableChatBotBanner(R.layout.my_chatbot_banner); uiConfigBuilder.build();</pre>

Handle Custom URLs in Chat

Have your agents pass along custom URLs to perform specific actions in your mobile app.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

When an agent sends a standard link to a user in your app, a preview tile appears that the user can tap to view in a browser. However, you can come up with your own URL scheme that displays a custom tile and performs a custom action within your app.

1. Create a custom URL scheme and expression.


Create a URL scheme using the [AppEventList](#) class. This class allows you to add URLs either for specific paths (such as `action/settings`) or using regular expressions (such as `action\\a.*`). Patterns are matched in the order that you add them to the [AppEventList](#) object. Be sure to add items starting with the most specific and ending with the most generic.

```
// Create an app event list
// TO DO: Replace "servicesdk" with your own unique scheme
AppEventList appEventList = new AppEventList("servicesdk");

// Add a regular expression for a path
appEventList.addDescriptionForPath("action/settings", "Tap for regex action");

// Add a custom path for a specific path
appEventList.addDescriptionForExpression("action\\a.*", "Tap for path action");
```

2. Implement the [AppLinkClickListener](#) so that you can handle when the user taps on the URL.

 **Note:** In order to minimize or maximize the chat window, you need access to the [ChatUIClient](#) that you used when creating the chat UI.

```
public static class ChatAppLinkClickListener implements AppLinkClickListener {

    private final Activity mActivity;
    private final ChatUIClient mUIClient;

    // Constructor requires the chat UI client and the activity
    public ChatAppLinkClickListener(ChatUIClient uiClient, Activity activity) {
        mActivity = activity;
    }
}
```

```

        mUIClient = uiClient;
    }

    @Override
    public void didReceiveAppEventWithURL(@NonNull String url) {

        if (url.contains("settings")) {

            // Minimize chat window
            mUIClient.minimize();

            // TO DO: Perform some action for this url. For example:
            Intent intent = new Intent(mActivity, ChatSettingsActivity.class);
            mActivity.startActivity(intent);
        }
        if (url.contains("alert")) {

            // TO DO: Perform some action for this url. For example:
            Toast.makeText(
                mActivity.getApplicationContext(),
                "An example of an alert",
                Toast.LENGTH_SHORT)
                .show();
        }

        // NOTE: You can maximize the chat window using
        //         mUIClient.maximize();
    }
}

```

3. Configure the chat SDK using the [ChatUIConfiguration](#) builder. Add the [AppEventList](#) object and the [AppLinkClickListener](#).

```

new ChatUIConfiguration.Builder()
    // ... other ChatUIConfiguration details
    .chatUIConfigurationBuilder.appEventList(appEventList)
    .appLinkClickListener(new ChatAppLinkClickListener(context))

```

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.



Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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:

```
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:

```
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:

```
// 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.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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)
    .maxLength(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)
    .maxLength(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("AHHHHHHHHH!!!", "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.

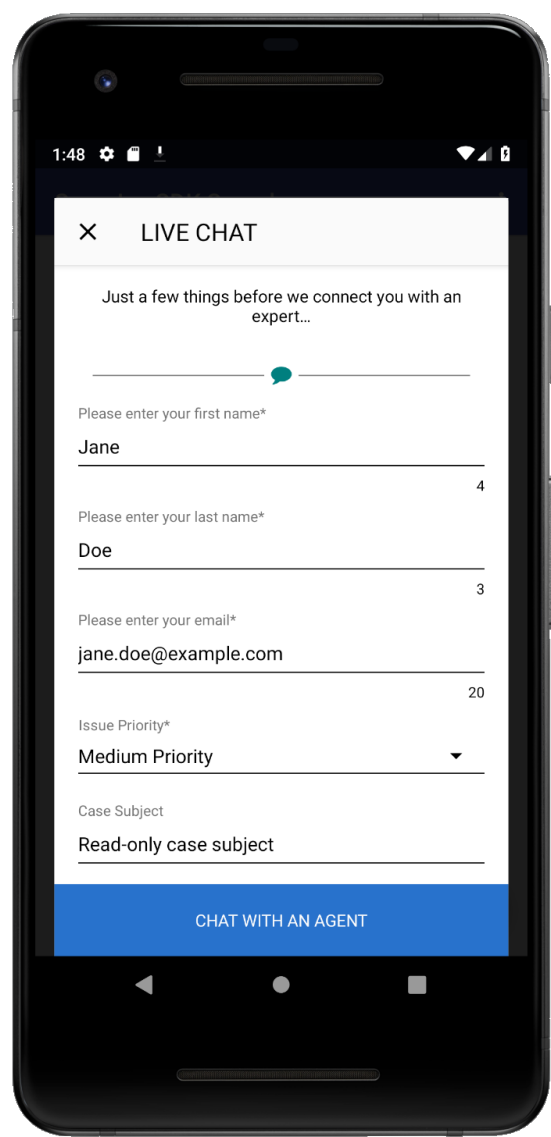
In Java:

```
// 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.




And the agent sees the following UI from the console.

* Chat - Visitor +			
<div>Live Agent Chat</div> <div>Visitor</div>			
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.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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 interaction
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



Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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.

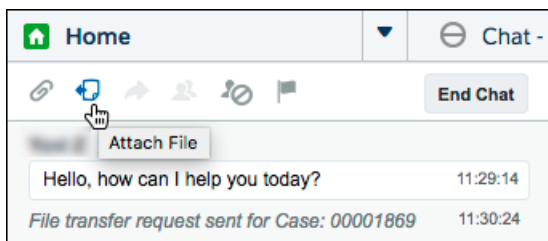
Transfer File to Agent

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



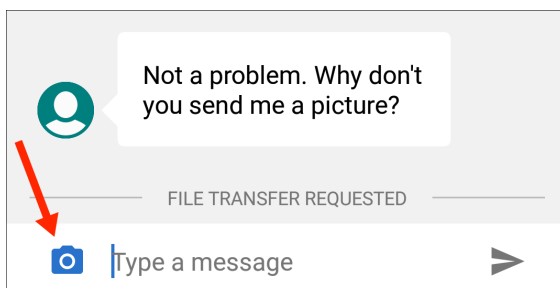
Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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



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.



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 54, 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.



Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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 Chat 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 Chat 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()`.

SDK Customizations with the Service Chat SDK for Android

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

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

[Customize Colors with the Service Chat 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 Chat 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.

[Customize Chat Images with the Service Chat SDK](#)

Specify custom images used throughout the chat UI.

Customize Colors with the Service Chat SDK

You can customize the look and feel of the interface by specifying the colors used throughout the UI.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

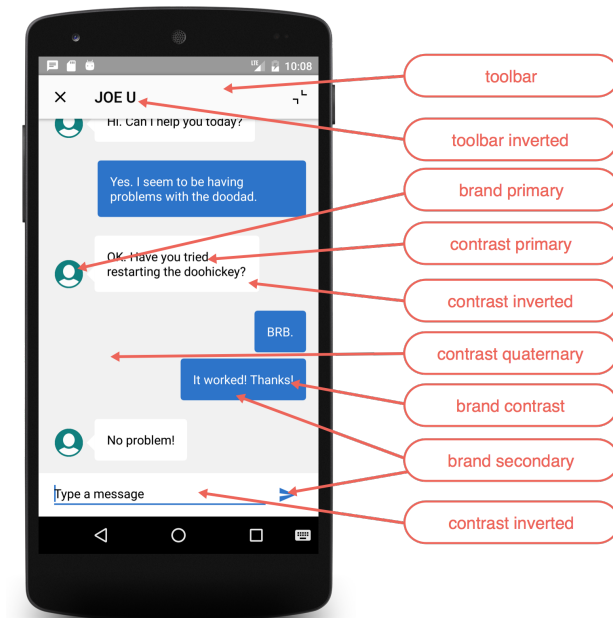
To customize the colors, create color resource values in your project's `colors.xml` file that correspond to the same resource names specified in this documentation.

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.

Chat UI Branding:



The following branding tokens are available for customization.

Token Name	Default Value	Description / Sample Uses
Toolbar salesforce_toolbar	#FAFAFA 	The toolbar background color.
Toolbar Inverted salesforce_toolbar_inverted	#010101 	Toolbar text and icon color.
Brand Primary salesforce_brand_primary	#007F7F 	The banner background color. Chat: Line under the message you're typing.
Brand Secondary salesforce_brand_secondary	#2872CC 	Chat: User text bubbles.
Brand Contrast salesforce_brand_contrast	#FCFCFC 	Title text color.
Contrast Primary salesforce_contrast_primary	#000000 	Primary body text color.
Contrast Secondary salesforce_contrast_secondary	#6D6D6D 	
Contrast Tertiary salesforce_contrast_tertiary	#BABABA 	
Contrast Quaternary salesforce_contrast_quaternary	#F1F1F1 	Chat: Background color for the chat feed screen.

Token Name	Default Value	Description / Sample Uses
Contrast Inverted <code>salesforce_contrast_inverted</code>	#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.
Feedback Primary <code>salesforce_feedback_primary</code>	#E74C3C 	Text color for error messages.
Feedback Secondary <code>salesforce_feedback_secondary</code>	#2ECC71 	
Feedback Tertiary <code>salesforce_feedback_tertiary</code>	#F5A623 	
Title Color <code>salesforce_title_color</code>	#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.
Overlay <code>salesforce_overlay</code>	#000000 (40% alpha)	

Customize and Localize Strings with the Service Chat 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.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

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](#)
- [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 1: Supported Languages

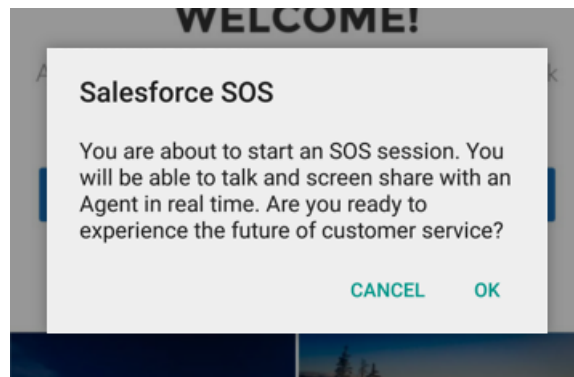
Language Code	Language
values-ar	Arabic
values-cs	Czech
values-da	Danish
values-de	German
values-el	Greek
values-en	English
values-es	Spanish
values-fi	Finnish
values-fr	French
values-hu	Hungarian
values-in	Indonesian
values-it	Italian
values-iw	Hebrew
values-ja	Japanese
values-ko	Korean
values-nl	Dutch
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
values-uk	Ukrainian

Language Code	Language
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:

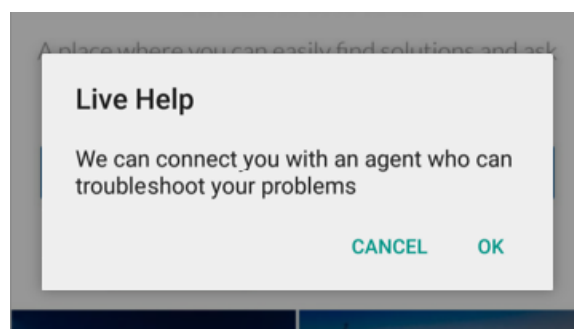


You can change the title and body of this dialog by changing the `sos_title` and the `sos_connect_prompt` strings in the `strings.xml` file in the `values` folders for your locale (`values-en/` for English):

```
<!-- 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:



Customize Chat Images with the Service Chat SDK

Specify custom images used throughout the chat UI.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

The Service Chat SDK allows you to customize chat images in the UI. If you want to change an image, override the `Drawable` for the image you want to customize using the exact same file name. The following table contains a list of the most common overridable images.

Image File Name	Description
chat_ic_footer_menu.xml	Hamburger menu on the bottom of the window, next to the chat input. (48dp x 48dp)
chat_ic_minimized_connecting.xml	Connecting icon when the view is minimized. (32dp x 29dp)
chat_minimized_message_indicator.xml	Chat bubble unread indicator when the view is minimized. (27dp x 24dp)
common_ic_close.xml	Close button at the top left of the window. (24dp x 24dp)
salesforce_agent_avatar.xml	Agent avatar used beside the agent's chat bubble. (36dp x 36dp)
salesforce_ic_message_send.xml	Send message button next to chat input. (24dp x 24dp)
salesforce_ic_minimize.xml	Minimize button at the top right of the window. (18dp x 18dp)

Note: To customize the chat bot avatar and the bot banner, see [Use Einstein Bots with Chat](#) on page 32.

For instance, if you'd like to override the agent avatar during a chat session, add a `Drawable` to your `res/drawables` folder with the name `salesforce_agent_avator.xml`. This value should contain image info about the agent avatar.

The following list contains *all* the customizable `Drawables` used throughout the chat UI: `chat_button.xml`, `chat_button_pressed.xml`, `chat_footer_menu_item.xml`, `chat_ic_bubble.xml`, `chat_ic_close.xml`, `chat_ic_collapse.xml`, `chat_ic_footer_menu.xml`, `chat_ic_last_photo.xml`, `chat_ic_minimized_connecting.xml`, `chat_ic_photo_gallery.xml`, `chat_menu_bottom_button.xml`, `chat_menu_button.xml`, `chat_menu_header.xml`, `chat_menu_solo_button.xml`, `chat_menu_speech_arrow.xml`, `chat_menu_top_button.xml`, `chat_minimized_message_indicator.xml`, `agent_initial_avatar.xml`, `link_preview_arrow.xml`, `progress_indeterminate_horizontal_material.xml`, `salesforce_agent_avatar.xml`, `salesforce_button.xml`, `salesforce_button_solid.xml`, `salesforce_horizontal_rule.xml`, `salesforce_ic_camera.xml`, `salesforce_ic_message_send.xml`, `salesforce_ic_minimize.xml`, `salesforce_loading_ball.xml`, `salesforce_message_bubble_overlay.xml`, `salesforce_message_bubble_received.xml`, `salesforce_message_bubble_received_speech_arrow.xml`, `salesforce_message_bubble_sent.xml`, `salesforce_minimized_view.xml`, `salesforce_minimized_view_toolbar.xml`, `vector_drawable_progress_indeterminate_horizontal.xml`.

Troubleshooting the Service Chat SDK

Get some guidance when you run into issues.

Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

[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 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 Chat SDK](#)

If you're trying to build a Service Chat SDK project that explicitly embeds the Salesforce Mobile SDK, exclude these two maven dependencies to prevent conflicts.

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 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.salesforce-scr.com/chat/rest/`, then use the following value in your code: `MyDomainName.my.salesforce-scr.com`.
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 Chat SDK

If you're trying to build a Service Chat 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:224.2.6')
{
    exclude group: 'com.salesforce.mobilesdk', module: 'SmartStore'
    exclude group: 'com.salesforce.mobilesdk', module: 'SalesforceSDK'
}
```

Data Protection and Security in the Service Chat SDK for Android


The Service Chat SDK does not collect or store personal data from its users. We ensure that data is secure both locally and when in transit.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

- **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.
- **Secure data in transit.** All network communication occurs over SSL using TLS 1.2.

Reference Documentation

Reference documentation for Service Chat SDK for Android.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

To access the reference documentation for the Service Chat 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.

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid

service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

Chat Index

- [AgentAvailabilityClient](#)
- [AgentListener](#)
- [AppEventList](#)
- [AppLinkClickListener](#)
- [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`

Common Index

- `Async`
- `AuthenticatedUser`
- `AuthTokenProvider`
- `PushNotificationListener`
 - `onPushNotificationReceived`
- `SalesforceConnectedApp`
- `ServiceAnalytics`
- `ServiceAnalyticsListener`
- `ServiceLogging`
 - `addSink`
 - `setLogLevel`
- `ServiceLoggingSink`

Resource Files

- Chat String Resources
- Common String Resources

Deprecated APIs

- `ArticleSummary`
 - `getArticleId`
 - `getArticleNumber`
- `DataCategorySummary`
 - `getName`
- `KnowledgeClient`
- `KnowledgeConfiguration`
 - `KnowledgeConfiguration.Builder`
 - `offlineResourceConfig`
- `KnowledgeCore`


- deleteCache
- KnowledgeCssProvider
- KnowledgeImageProvider
 - getImageForArticle
 - getImageForDataCategory
- KnowledgeJsProvider
- KnowledgeUI
 - configure
 - createClient
- KnowledgeUIClient
 - KnowledgeUIClient.OnCloseListener
 - launchHome
 - setOnCloseListener
- KnowledgeUIAnalytics
- KnowledgeUIConfiguration
- KnowledgeViewAddition
 - createView
 - getEnterAnimator
 - getExitAnimator
 - initView
 - visibleFor
- OfflineResourceConfig
- CaseClient
 - clearCache
- CaseClientCallbacks
 - getHiddenFields
- CaseConfiguration
 - CaseConfiguration.Builder
 - caseListName
 - enablePush
- CasesUIAnalytics
- CaseUI
- CaseUIClient
 - getCoreClient
 - launch
 - launchCaseList

- `launchCasePublisher`
- `CaseUIConfiguration`
- `AudioStats`
- `AutoCompleteTextView`
- `EditText`
- `SosAvailability`
 - `SosAvailability.Listener`
- `SosAVListener`
- `SosMaskedField`
 - `setMask`
 - `showMask`
 - `useFocusMasking`
- `MultiAutoCompleteTextView`
- `Sos`
 - `hideKeyboard`
 - `getHoldState`
 - `getState`
 - `Sos.SessionBuilder`
 - `start`
 - `showKeyboard`
- `SosConfiguration`
 - `SosConfiguration.Builder`
 - `connectingUi`
 - `disableToasts`
- `SosConnectionListener`
- `SosHoldListener`
- `SosListener`
- `SosMaskingListener`
- `SosNetworkStatsListener`
- `SosNetworkTestListener`
- `SosOptions`
- `SosShareTypeListener`
- `SosState`
- `SosToast`
- `TextView`
- `VideoStats`
- `View`

- [Knowledge String Resources](#)
- [Case Management String Resources](#)
- [SOS String Resources](#)

Additional Resources

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

 **Important:** The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging for In-App and Web](#) before that date. Messaging offers many of the [chat features that you](#) love plus asynchronous conversations that can be picked back up at any time. Learn about chat retirement in [Help](#).

- Embedded Service Chat SDK for Mobile Apps Resources
 - [Service Chat SDK Developer Center](#)
 - [Service Chat 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

A

- analytics [23](#)
- authentication [16](#)
- authentication using mobile sdk [17](#)

B

- branding [56](#)

C

- color branding [57](#)
- customize colors [57](#)

D

- data protection [64](#)

E

- Einstein bots [32](#)

F

- file transfer [51](#)

L

- localization [59](#)
- logging in Android [63](#)

N

- notifications [22](#)

P

- prerequisites [13](#)
- push notifications [22](#)

R

- reference index [64](#)
- reference overview [64](#)
- release notes [2](#)

S

- SDK prerequisites [13](#)
- sdk setup [12](#)
- service cloud setup [2](#)
- setup [2](#), [12](#)
- strings [59](#)

T

- transfer file [51](#)
- troubleshooting
 - compile [63](#)
- tutorials [25](#)

U

- ui customization [56](#)