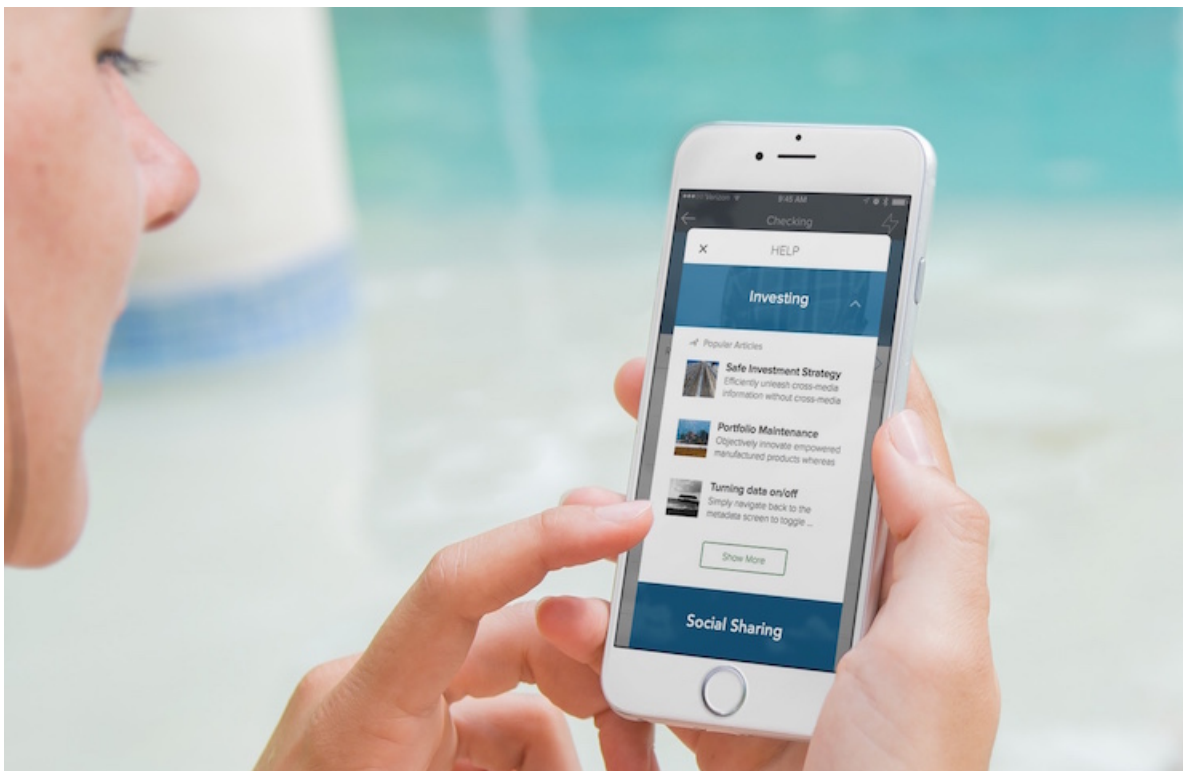




Embedded Service Chat SDK for iOS

Developer Guide

Version 238.0



CONTENTS

- [Embedded Service Chat SDK for iOS Developer Guide](#) 1
- Release Notes 2
- Service Cloud Setup 2
- SDK Setup 12
- iOS Tutorials & Examples 24
 - Chat 30
- SDK Customizations 64
- Troubleshooting 74
- Data Protection and Security 75
- Reference Documentation 76
- Additional Resources 82
- [Index](#) 83

EMBEDDED SERVICE CHAT SDK FOR IOS 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 246.0.0)

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

Component	Version Number
Chat	4.1.2
ServiceCore	4.2.8



Note: If you are using Xcode 12.3 or later, you must install the SDK using XCFramework files. To learn more, review the [installation instructions](#) on page 15.

[Release Notes](#)

Check out the new features and known issues for the iOS 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.

[iOS Tutorials & Examples](#)

Get going quickly with these short introductory tutorials.

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

Add the Chat experience to your mobile app.

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

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

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 iOS.

[Additional Resources](#)

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

SEE ALSO:

[Service SDK for iOS Release Notes](#)

[Service SDK for iOS Reference Documentation](#)

[Service SDK for Android Developer Guide](#)

Release Notes

Check out the new features and known issues for the iOS 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 iOS, visit github.com/forcedotcom/ServiceSDK-iOS/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.



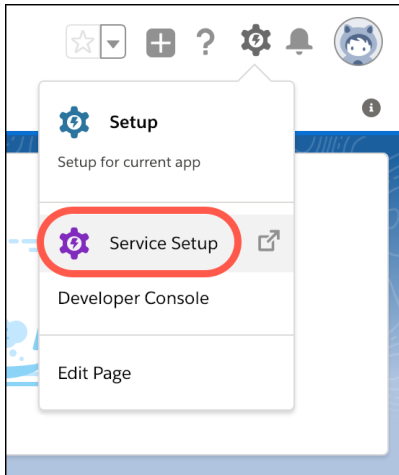
Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging 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](#).

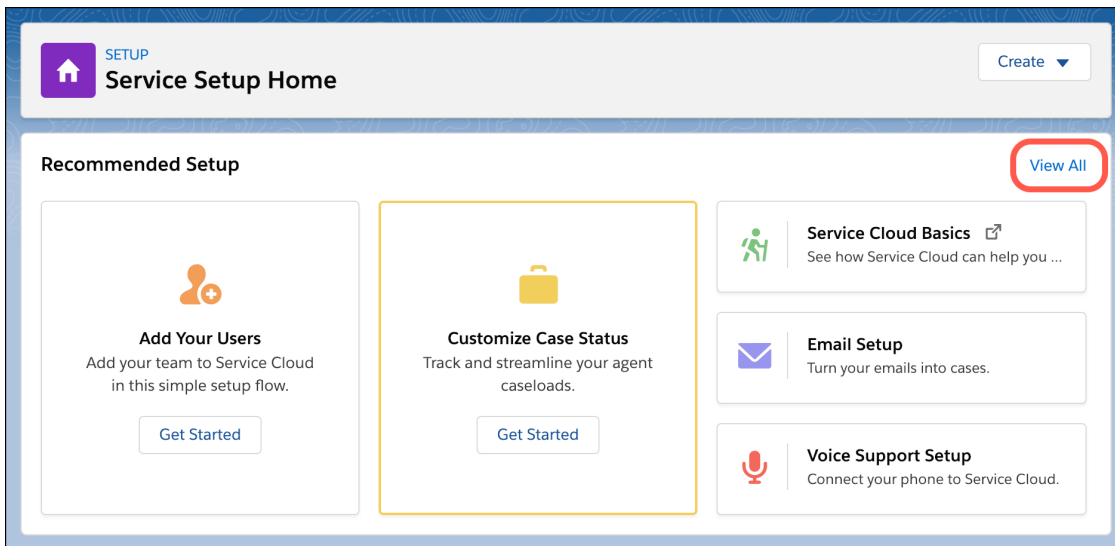
 **Warning:** If you use the [Permitted Domains setting in your Chat deployment](#), you'll get unreliable information from the chat availability check in the SDK. For instance, the agent availability status may always return `false`. If you want to use Permitted Domains for your web chat deployment,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

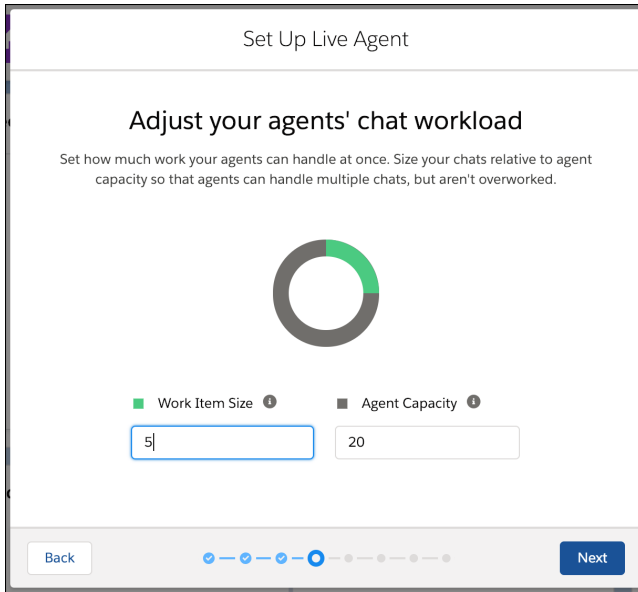
Name

Priority

Back

Next

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

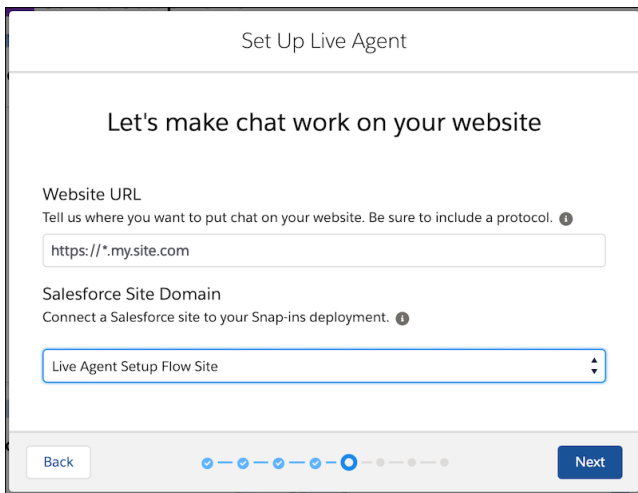


The screenshot shows the 'Set Up Live Agent' screen with the heading 'Adjust your agents' chat workload'. Below the heading is a sub-header: 'Set how much work your agents can handle at once. Size your chats relative to agent capacity so that agents can handle multiple chats, but aren't overworked.' 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:

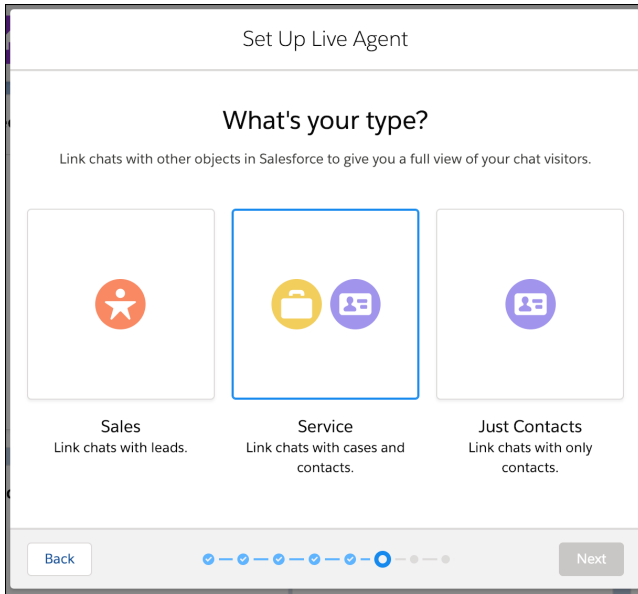
- The URL of your site.
- `https://`, followed by the last part of your site's URL: `https://*.my.site.com`, `https://*.salesforce-sites.com`, or `https://*.force.com`.

Then create or select a site.

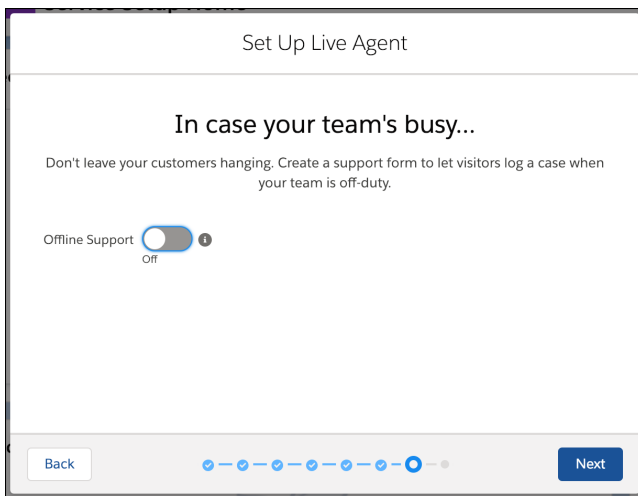


The screenshot shows the 'Set Up Live Agent' screen with the heading 'Let's make chat work on your website'. Below the heading are two sections: 'Website URL' with the instruction 'Tell us where you want to put chat on your website. Be sure to include a protocol.' and a text input field containing 'https://*.my.site.com'; 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.

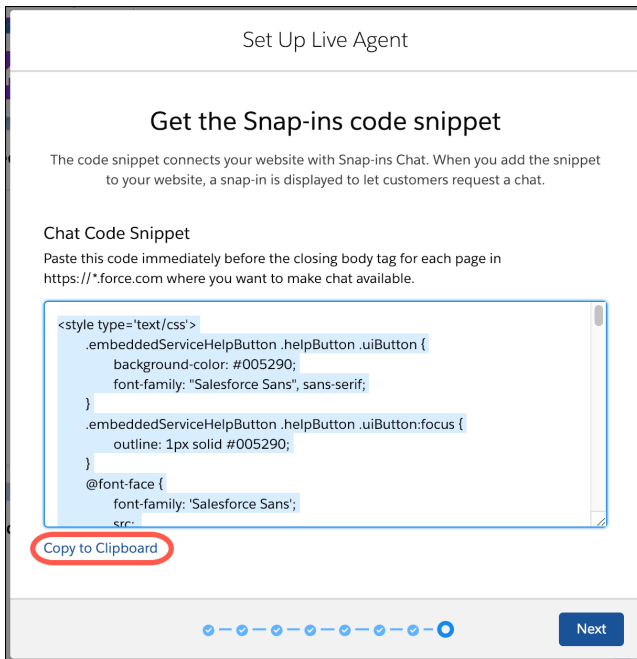
9. For the type of chat, select **Service**.



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
    }
});

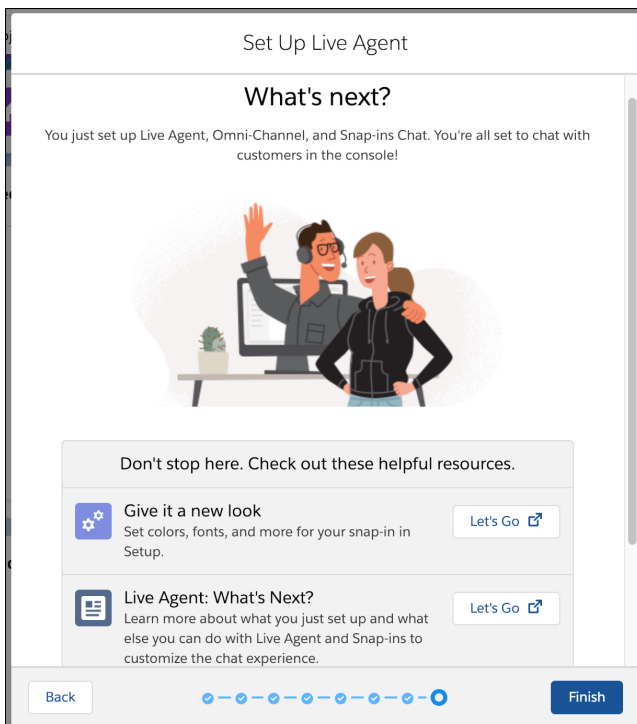
```

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](#).

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



- (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

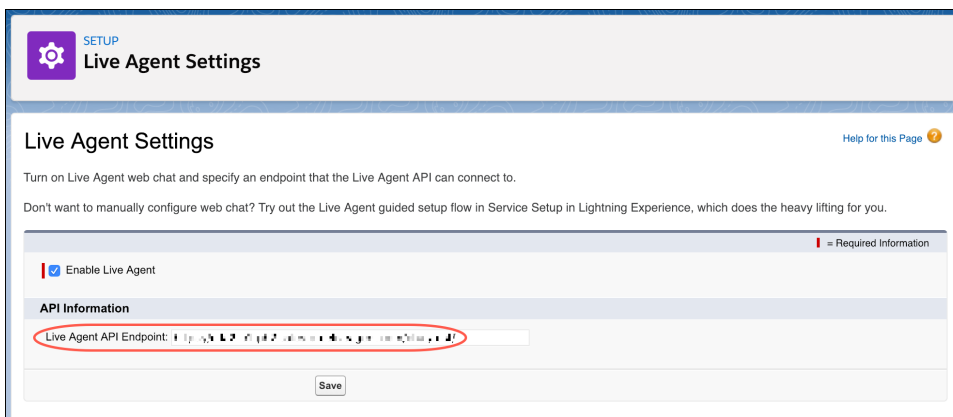
After you've set up chat in the console, supply your app developer with four values: the chat endpoint hostname, the organization ID, the deployment ID, and the button ID. You can get this information from your org's setup.

 **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:

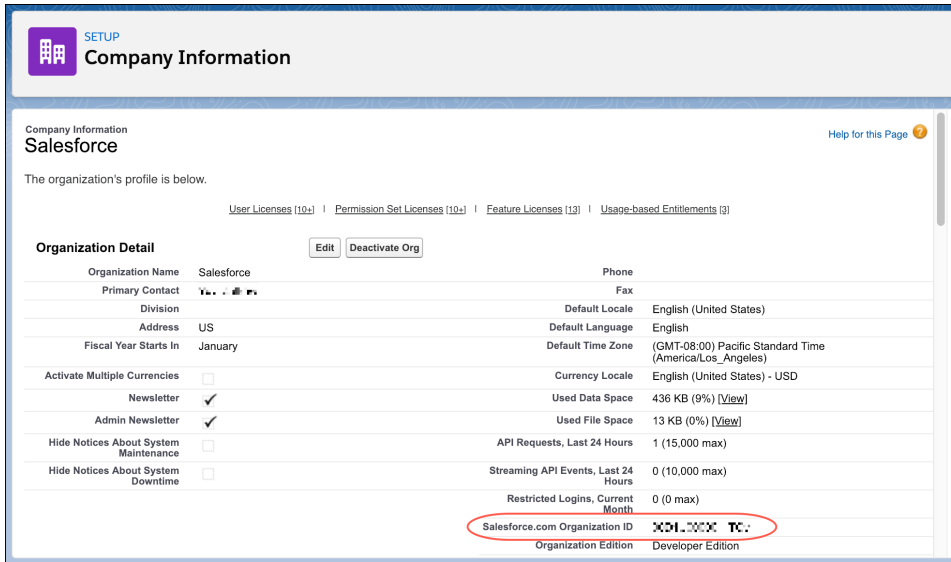
```
https://d.gla5.gus.salesforce.com/chat/rest/
```

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.

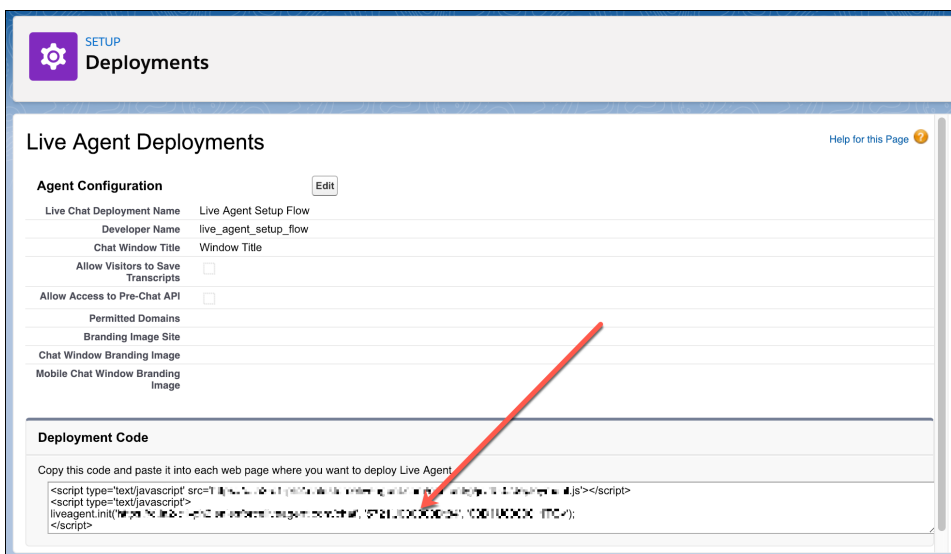
User Licenses (10+) | Permission Set Licenses (10+) | Feature Licenses (13) | Usage-based Entitlements (3)

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	<input type="checkbox"/>
Allow Access to Pre-Chat API	<input type="checkbox"/>
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>
```

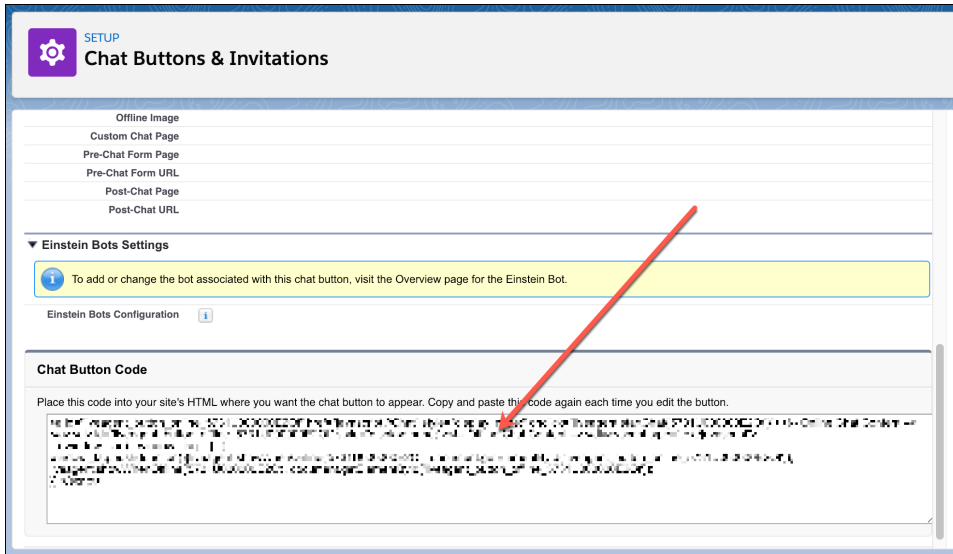
The deployment ID value is:

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 iOS](#)

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

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

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 Chat SDK for iOS](#)

Before you can use the iOS SDK, install the SDK and configure your project.

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

The Service Chat SDK provides an authentication mechanism that allows your users to access user-specific information in Service Cloud. To authenticate, create an `SCSAuthenticationSettings` object and pass it to the SDK.

[Notifications with the Service Chat SDK for iOS](#)


The Service Chat SDK can display notifications for activity related to Chat.

[Prepare Your App for Submission](#)

If you're not using Swift Package Manager to install the SDK, you need to strip development resources (such as unneeded architectures and header resources) from the Service Chat SDK before you can submit your app to the App Store.

Requirements for the Service Chat SDK for iOS

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 Development Requirements

To develop using this SDK, you must have [iOS SDK](#) version 12 or later.

 **Important:** Apple mandates a privacy manifest for third-party SDKs that have a large user base. Service Chat SDK is excluded from Apple's list of third-party SDK requirements because it has a small user base compared to popular third-party SDKs. Also, the SDK has reached the end-of-sale state, and end-of-sale SDKs haven't been added to the requirements. So, Service Chat doesn't require a privacy manifest. See [Upcoming third-party SDK requirements](#) in *Apple Developer Support*.


Service Chat also doesn't collect or store user data locally in the SDK and doesn't track data. However, if you use a pre-chat form and configure it to store any data entered by the user in Salesforce, add a privacy manifest with this information to the host app.

Mobile App Requirements

Any app that uses this SDK requires [iOS](#) version 12 or later.

Accessibility with the Service Chat SDK for iOS

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 creating the `SCSChatConfiguration` object, set `allowMinimization` and `defaultToMinimized` to `false`.

```
let config = SCSChatConfiguration(liveAgentPod: "YOUR_POD_NAME",
                                  orgId: "YOUR_ORG_ID",
                                  deploymentId: "YOUR_DEPLOYMENT_ID",
                                  buttonId: "YOUR_BUTTON_ID")

config?.allowMinimization = false
config?.defaultToMinimized = false
```

See [Configure a Chat Session](#).

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](#).

Dynamic Text Warning

When changing the iOS text size, the Apple Accessibility Inspector displays the following warning: "Dynamic Text font sizes are unsupported." Dynamic text is supported, but it requires restarting the app after changing the font size.

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 Chat SDK for iOS

Before you can use the iOS SDK, install the SDK and configure your project.


 **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. Add the SDK frameworks to your project.

You can add the frameworks using [Swift Package Manager](#), using [CocoaPods](#), or by downloading and adding the frameworks manually.

To add the frameworks using Swift Package Manager, add a package dependency to your app using the following public URL:
`https://github.com/Salesforce-Async-Messaging/Swift-Package-ServiceChat.`

To learn more about package dependencies, see [Adding Package Dependencies to Your App](#) in Apple's Developer Documentation.

 **Important:** The Swift Package Manager installation method is only supported for the Service Chat feature. To install the SDK for Knowledge or Case Management, use one of the other installation methods.

Alternatively, you can [add the frameworks files manually](#) or [install the frameworks using CocoaPods](#).

2. Add descriptions for why the app needs access to the device's camera and photo library.

Add string values for "Privacy - Camera Usage Description" and "Privacy - Photo Library Usage Description" in your `Info.plist` file. To learn more about these properties, see [Cocoa Keys](#) in Apple's reference documentation.

Sample values for these keys:

```
<key>NSCameraUsageDescription</key>
<string>Used when sending an image to an agent.</string>
```

```
<key>NSPhotoLibraryUsageDescription</key>
<string>Used when sending an image to an agent.</string>
```

You're now ready to get started using the SDK!

[Add the Service SDK Frameworks with CocoaPods](#)

Add the SDK frameworks using CocoaPods, a developer tool that automatically manages dependencies.

[Add the Service SDK Frameworks Manually](#)

Add the SDK frameworks by manually embedding the appropriate frameworks.

Add the Service SDK Frameworks with CocoaPods

Add the SDK frameworks using CocoaPods, a developer tool that automatically manages dependencies.



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. If you haven't already done so, install the [CocoaPods](#) gem and initialize the CocoaPods main repository.

```
sudo gem install cocoapods
pod setup
```



Note: The minimum supported version of CocoaPods is 1.0.1. If you're not sure what version you have, use `pod --version` to check the version number.

2. If you already have [CocoaPods](#) installed, update your pods to the latest version.

```
pod update
```

3. Change to the root directory of your application project.
4. Create or edit a file named `Podfile` that contains the Service Chat SDK dependency.
 - a. If you want to install the complete Service Chat SDK, update your `Podfile` to include `ServiceSDK`.

```
source 'https://github.com/CocoaPods/Specs.git'
source 'https://github.com/goinstant/pods-specs-public'

# To use the SDK (with all components)
target '<your app target>' do
  pod 'ServiceSDK'
end
```

- b. If you want to install a single Service Chat SDK component, create a similar `Podfile` to the one specified above, but only include the desired pod.

Feature	Pod name
Chat	ServiceSDK/Chat

For example, the following `Podfile` installs Chat.

```
source 'https://github.com/CocoaPods/Specs.git'
source 'https://github.com/goinstant/pods-specs-public'


# To use Chat
target '<your app target>' do
  pod 'ServiceSDK/Chat'
end
```

If you don't specify a version number, you automatically get the latest version of that component. If you want to add a specific version to your component, be sure to add the version number of the Service Chat SDK and *not* the version number of the individual component.


5. Run the CocoaPods installer.

```
pod install
```

This command generates a `.xcworkspace` file for you with all the dependencies included.

 **Note:** If you run into issues installing or updating the SDK, clear the local CocoaPods cache and then perform a `pod update`.


6. Open the `.xcworkspace` file that CocoaPods generated and continue with the installation process.


 **Note:** Be sure to open the `.xcworkspace` file (which includes all the dependencies) and *not* the `.xcodeproj` file.

Once you've added the SDK frameworks, proceed with [the installation instructions](#) on page 15.

Add the Service SDK Frameworks Manually

Add the SDK frameworks by manually embedding the appropriate frameworks.

 **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 you are using Xcode 12.3 or later, you must install the SDK using XCFramework files.

1. Download the SDK frameworks from the [Service Chat SDK download page](#).
2. Embed the relevant Service Chat SDK frameworks into your project.

You can find the framework files within the `Frameworks` folder. Specifically, the following frameworks are available for you to use:

Framework	Description
ServiceCore	Contains all the common components used by the Service SDK.
ServiceChat	Contains access to the Chat features of the SDK.


Add the relevant frameworks to the **Frameworks, Libraries, and Embedded Content** section of the **General** tab for your target app.

 **Important:** Before embedding a framework into your project, be sure to copy the framework file into your project folder (or another location found in the framework search path). If Xcode can't find the framework, you won't be able to access its contents in your code.

Once you've added the SDK frameworks, proceed with [the installation instructions](#) on page 15.

Authentication with the Service Chat SDK for iOS

The Service Chat SDK provides an authentication mechanism that allows your users to access user-specific information in Service Cloud. To authenticate, create an `SCSAuthenticationSettings` object and pass it 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](#).

Authentication Settings

Authentication with the Service Chat SDK uses an `SCSAuthenticationSettings` object. Create this object with a client ID and a dictionary containing authentication settings. This authentication settings dictionary must contain a URL for your org (`SCSAuth2JSONKeyInstanceUrl`) and an access token (`SCSAuth2JSONKeyAccessToken`). If your OAuth2 flow supports refresh tokens, include a refresh token (`SCSAuth2JSONKeyRefreshToken`) to the authentication settings.

In Swift:

```
// Specify auth info
let myClientId: String = "CLIENT_ID_VALUE"
let authDictionary: [SCSAuth2JSONKey: String] =
    [ .instanceUrl : "https://MyDomainName.my.salesforce.com",
      .accessToken : "ACCESS_TOKEN_VALUE" ]

// Create auth settings object
let authSettings = SCSAuthenticationSettings(oauth2: authDictionary,
                                             clientId: myClientId)
```

In Objective-C:

```
// Specify auth info
NSString *myClientId = @"CLIENT_ID_VALUE";
NSDictionary<SCSAuth2JSONKey, NSString*> *authDictionary =
    @{ SCSAuth2JSONKeyInstanceUrl : @"https://URL_FOR_YOUR_ORG.com",
      SCSAuth2JSONKeyAccessToken : @"ACCESS_TOKEN_VALUE" };

// Create auth settings object
SCSAuthenticationSettings *authSettings =
    [[SCSAuthenticationSettings alloc] initWithOAuth2Dictionary: authDictionary
                                                         clientId: myClientId];
```

If you're using the [Salesforce Mobile SDK](#), we provide a helper method that allows you to construct an [SCSAuthenticationSettings](#) object directly from the Mobile SDK user account. You can use this sample code after you've successfully logged in a user.

In Swift:

```
// Get user account info from the Salesforce Mobile SDK
let identity: SFUserAccountIdentity =
    SFUserAccountIdentity(userId: myUserId, orgId: myOrgId)
let account: SFUserAccount =
    SFUserAccountManager.sharedInstance().userAccount(forUserIdentity: identity)!

// Create auth settings object from SFUserAccount
let authSettings = SCSAuthenticationSettings(mobileSDK: account)
```

In Objective-C:

```
// Get user account info from the Salesforce Mobile SDK
SFUserAccountIdentity *identity =
    [SFUserAccountIdentityClass identityWithUserId:myUserId orgId:myOrgId];
SFUserAccount *account =
    [[SFUserAccountManagerClass sharedInstance] userAccountForUserIdentity:identity];

// Create auth settings object from SFUserAccount
SCSAuthenticationSettings *authSettings =
    [[SCSAuthenticationSettings alloc] initWithMobileSDKAccount:account];
```



Note: For developers who plan to use the [Salesforce Mobile SDK](#) for authentication, the [Mobile SDK Developer's Guide](#) contains [authentication](#) instructions. 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](#)). This documentation describes how to use the `SFDCOAuthLoginHost` property in your `info.plist` file to create a custom login URI.

If you plan to use remote push notification to alert the user when an event occurs in your org, call `registerForPushNotifications` on the [SCSAuthenticationSettings](#) object. To learn more, see [Notifications with the Service Chat SDK for iOS](#).

However you create an [SCSAuthenticationSettings](#) object, pass it to the Service Chat SDK during the authentication flow.

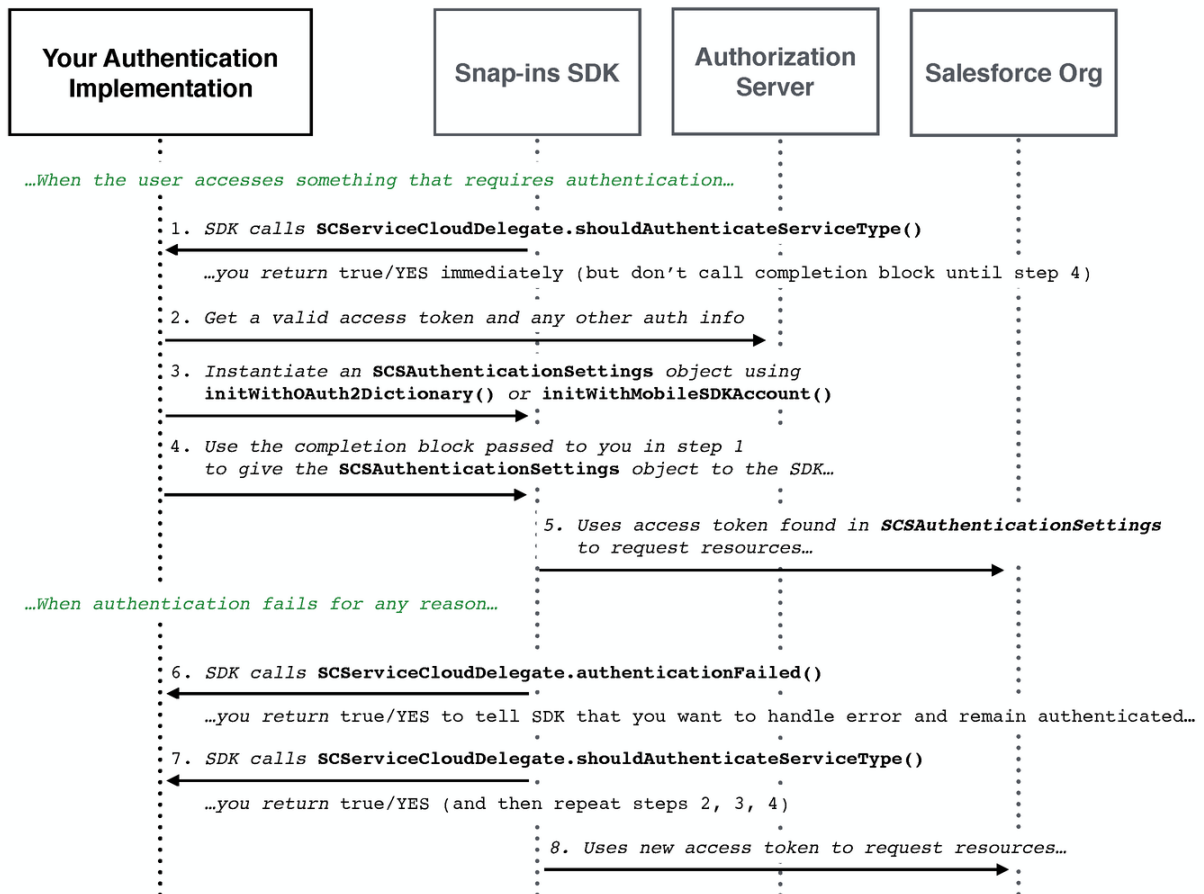
Authentication Flow

You can either authenticate on-demand when the SDK calls `serviceCloud(shouldAuthenticateServiceType:completion:)` in your `SCServiceCloudDelegate` implementation, or you can authenticate explicitly (that is, before the app attempts to show the relevant UI) using the `setAuthenticationSettings(settings:forServiceType:completion:)` method in the `ServiceCloud` shared instance.

With on-demand authentication, you perform the authentication asynchronously, after the SDK calls your `serviceCloud(shouldAuthenticateServiceType:completion:)` delegate method. Once authenticated, pass the [SCSAuthenticationSettings](#) object to the completion block that you're given in the `serviceCloud(shouldAuthenticateServiceType:completion:)` method.

The following sequence diagram illustrates the basic authentication flow for on-demand authentication.

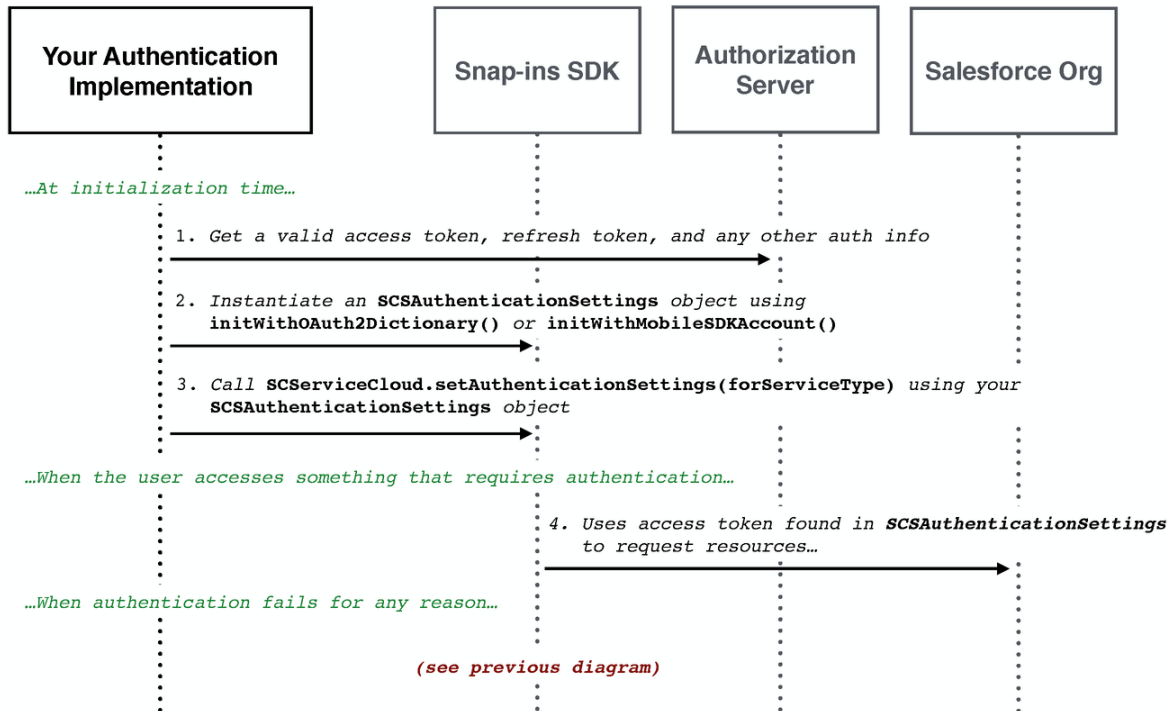
On-Demand Authentication



Alternatively, you can explicitly authenticate before any UI appears that requires authentication. Call the `setAuthenticationSettings(settings:forServiceType:completion:)` method in the `ServiceCloud` shared instance using the `SCSAuthenticationSettings` object.

The following sequence diagram illustrates the authentication flow for explicit authentication.

Explicit Authentication



To programmatically log out a user, call `setAuthenticationSettings(settings:forServiceType:completion:)` using `nil` for the `SCSAuthenticationSettings` argument.

Error Conditions

Implement `serviceCloud(authenticationFailed:forServiceType:)` in your `SCServiceCloudDelegate` object to handle error conditions. The SDK calls this method if the access token expires, and for any other scenario that results in an authentication failure. If you return `true` or `YES`, the SDK assumes that you want to proceed, and it subsequently calls `serviceCloud(shouldAuthenticateServiceType:completion:)` to give you a chance to send updated authentication information. If you return `false` or `NO`, the SDK goes back to the guest user state.

Sample Code

The following sample code illustrates how to implement an `SCServiceCloudDelegate` object to handle authentication.

```
class MyAuthHandler: NSObject, SCServiceCloudDelegate {

    override init() {
        super.init()

        // Subscribe to ServiceCloud events
        ServiceCloud.shared().delegate = self
    }
}
```

```

/**
 Implementation of a `ServiceCloudDelegate` method that allows you to
 authenticate for a given service.
 */
func serviceCloud(_ serviceCloud: ServiceCloud,
                  shouldAuthenticateServiceType service: SCServiceType,
                  completion: @escaping (SCSAAuthenticationSettings?) -> Void) -> Bool {

    // Rather than scrutinize the service to see if we want to authenticate,
    // let's just assume that we always want to authenticate...

    // TO DO: Authenticate asynchronously
    let urlRequest = URLRequest.init(url: URL(string: "https://example.com/auth")!)
    URLSession.shared.dataTask(with: urlRequest) { (data, response, error) in

        // TO DO: Populate the `SCSAAuthenticationSettings` object from the result.
        var authSettings: SCSAAuthenticationSettings?

        // Call the completion block with the authentication settings (asynchronously)
        completion(authSettings)

    }.resume()

    // Tell the SDK that we do plan to authenticate
    return true
}

/**
 Implementation of a `ServiceCloudDelegate` method to handle authentication
 failure events.
 */
func serviceCloud(_ serviceCloud: ServiceCloud,
                  authenticationFailed error: Error,
                  forServiceType service: SCServiceType) -> Bool {

    // For this example, let's not bother handling the error,
    // and just fall back to the guest user state...
    // TO DO: In your code, you should inspect this error.
    // If you want to handle the error, you could
    // return `true` and then you'd be called back in the
    // `shouldAuthenticateServiceType` method above.

    return false
}
}

```

Notifications with the Service Chat SDK for iOS

The Service Chat SDK can display notifications for activity related to Chat.




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 [Notifications for Chat Activity](#).

Prepare Your App for Submission


If you're not using Swift Package Manager to install the SDK, you need to strip development resources (such as unneeded architectures and header resources) from the Service Chat SDK before you can submit your app to the App Store.

 **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](#).

Xcode doesn't automatically strip unneeded architectures from dynamic libraries, nor remove some header and utility resources. Apps that don't do this clean up are rejected from the App Store. For example, you might receive the following error from iTunes Connect:

```
ERROR ITMS-90085:  
No architectures in the binary. Lipo failed to detect any architectures in the bundle  
executable.
```

You can resolve this problem by using the script provided in the `ServiceCore` framework that automatically strips unneeded architectures from the dynamic libraries and then re-signs them.

 **Note:** This process is not required if you used Swift Package Manager to install your app.

To use this script:

1. Select `Build Phases` for your project target.
2. Create a `Run Script` phase to run the script.

Access the `prepare-framework` script from within the `ServiceCore` framework in your project directory.

- If you installed the XCFramework files manually, use:


```
$SRCROOT/ServiceCore.xcframework/ios-arm64/ServiceCore.framework/prepare-framework
```

- If you installed the SDK with CocoaPods, use:

```
$PODS_ROOT/ServiceSDK/Frameworks/ServiceCore.framework/prepare-framework
```

- If you installed the standard framework files with version 224.0.2 or earlier, use:

```
$SRCROOT/ServiceCore.framework/prepare-framework
```

 **Important:** Verify that this script is in the *exact location* that you specify in this build phase.

This build phase must occur **after** the link phase and all embed phases. If you're using CocoaPods, make sure to put this script after the "[CP] Embed Pods Frameworks" phase.

iOS Tutorials & Examples

Get going quickly with these short introductory tutorials.

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](#).

In addition to these tutorials, check out our [GitHub repository \(github.com/forcedotcom/ServiceSDK-iOS\)](https://github.com/forcedotcom/ServiceSDK-iOS) for sample apps.

[Get Started with Chat](#)

Get rolling quickly with chat sessions between your customers and your agents.

Get Started with Chat

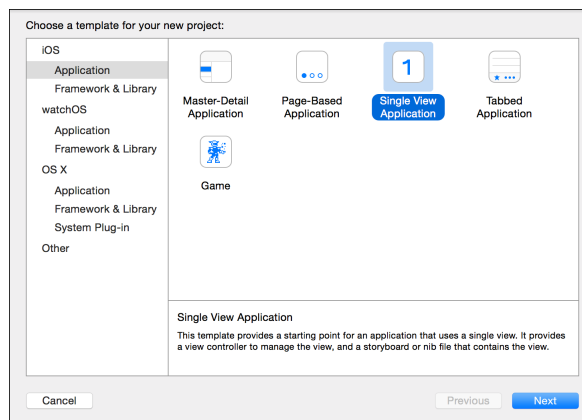
Get rolling quickly with chat sessions between your customers and your agents.

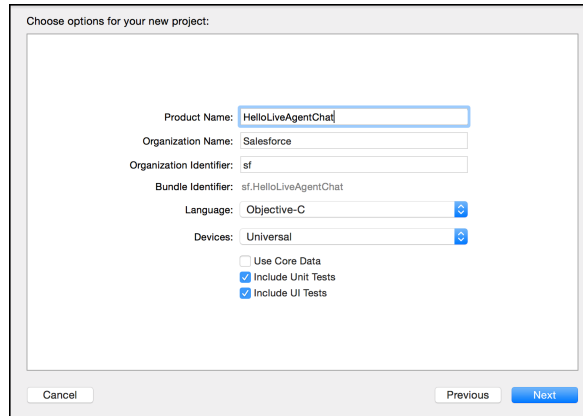
Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging 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 doing this tutorial, be sure that you've set up Service Cloud for Chat. See [Org Setup for Chat in Lightning Experience with a Guided Flow](#) for more information.

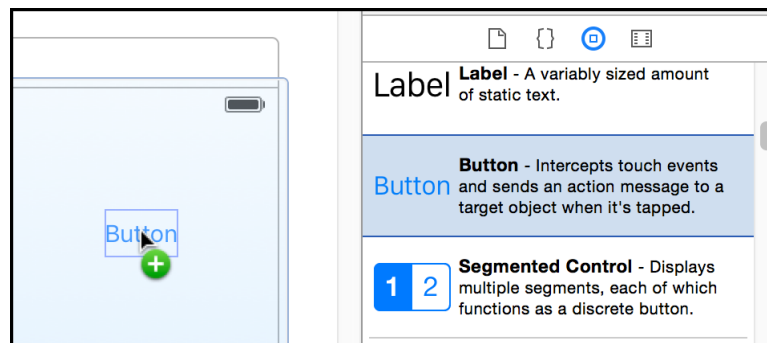
This tutorial shows you how to get Chat into your iOS app.

1. Create an Xcode project. For this example, let's make a Single View Application. Name it `HelloChat`.

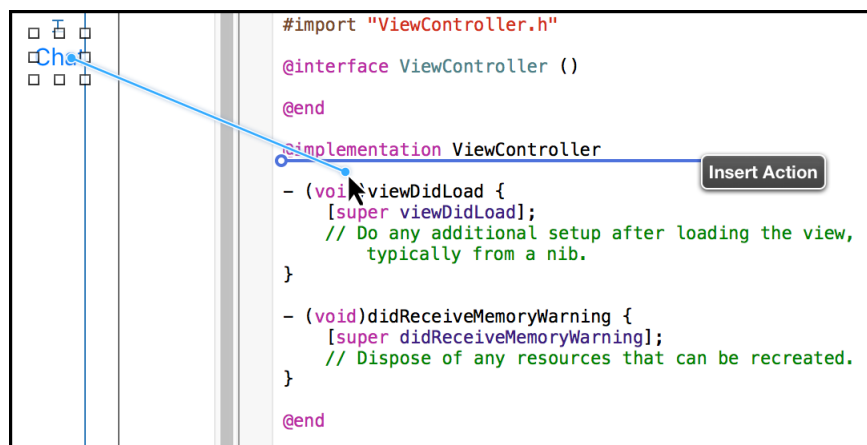


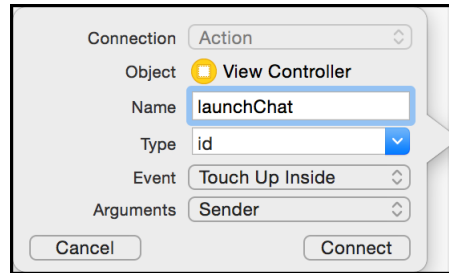


2. Install the SDK as described in [Install the Service Chat SDK for iOS](#).
3. Go to your storyboard and place a button somewhere on the view. Name it Chat.



4. Add a Touch Up Inside action to your UIViewController implementation. Name it launchChat.





5. Import the SDK. Wherever you intend to use the Chat SDK, be sure to import the ServiceCore framework and the ServiceChat framework.

In Swift:

```
import ServiceCore
import ServiceChat
```

In Objective-C:

```
@import ServiceCore;
#import ServiceChat;
```

6. Launch a chat session from within the `launchChat` method.

From the button action implementation, launch chat using the `showChat(with:showPrechat:)` method on `SCSChatInterface`.

In Swift:

```
@IBAction func launchChat(sender: AnyObject) {

    // Create configuration object
    if let config = SCSChatConfiguration(liveAgentPod: "YOUR_POD_NAME",
                                         orgId: "YOUR_ORG_ID",
                                         deploymentId: "YOUR_DEPLOYMENT_ID",
                                         buttonId: "YOUR_BUTTON_ID") {

        // Start the session
        ServiceCloud.shared().chatUI.showChat(with: config)
    }
}
```

In Objective-C:

```
- (IBAction)launchChat:(id)sender {

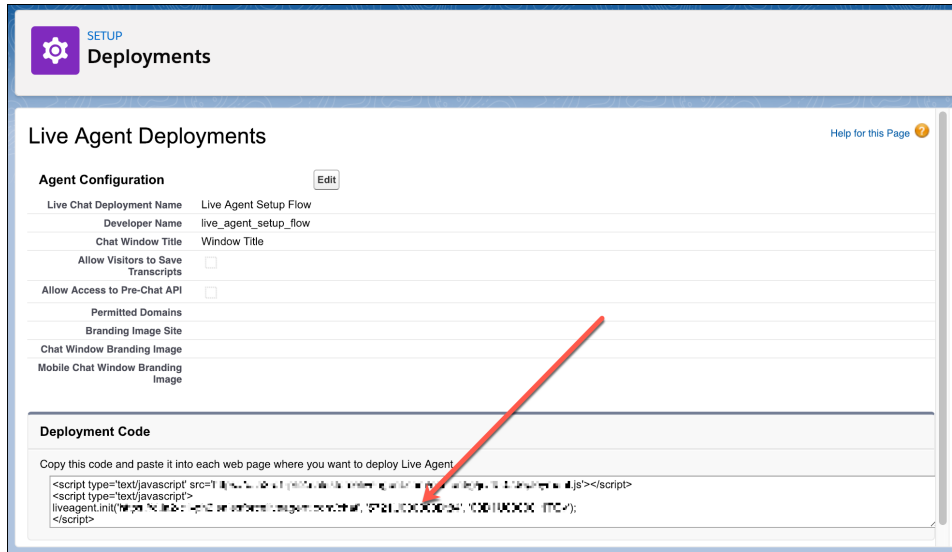
    SCSChatConfiguration *config =
        [[SCSChatConfiguration alloc] initWithLiveAgentPod:@"YOUR_POD_NAME"
                                                         orgId:@"YOUR_ORG_ID"
                                                         deploymentId:@"YOUR_DEPLOYMENT_ID"
                                                         buttonId:@"YOUR_BUTTON_ID"];

    // Start the session
    [[SCServiceCloud sharedInstance].chatUI showChatWithConfiguration:config];
}
```

Fill in the placeholder text for the Chat API endpoint, the org ID, the deployment ID, and the button ID.

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.



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

```
<script type='text/javascript'
src='https://d.gla3.gus.salesforce.com/content/g/js/44.0/deployment.js'></script>
<script type='text/javascript'>
liveagent.init('https://d.gla5.gus.salesforce.com/chat', '573B00000005KXz',
'00DB00000003Rxx');
</script>
```

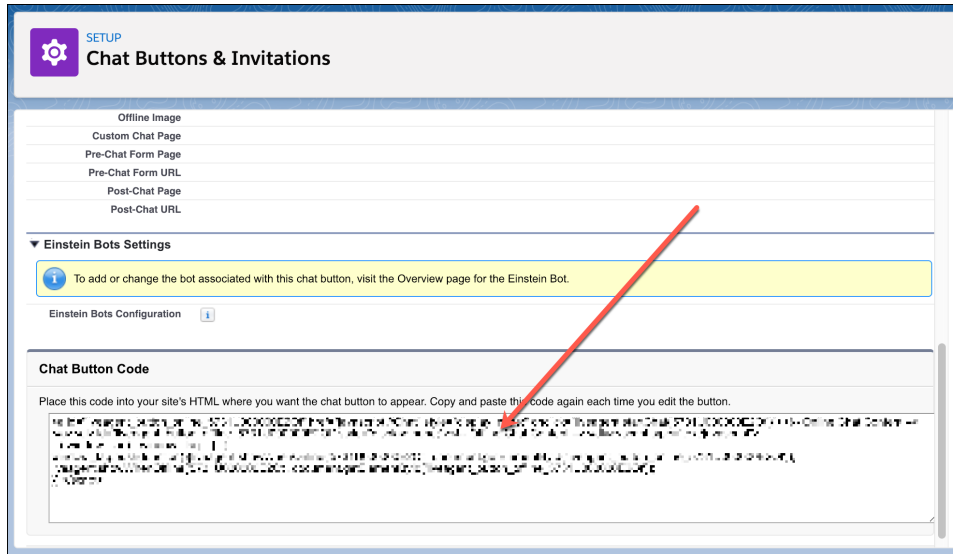
The deployment ID value is:

```
573B00000005KXz
```

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

Button ID

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



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

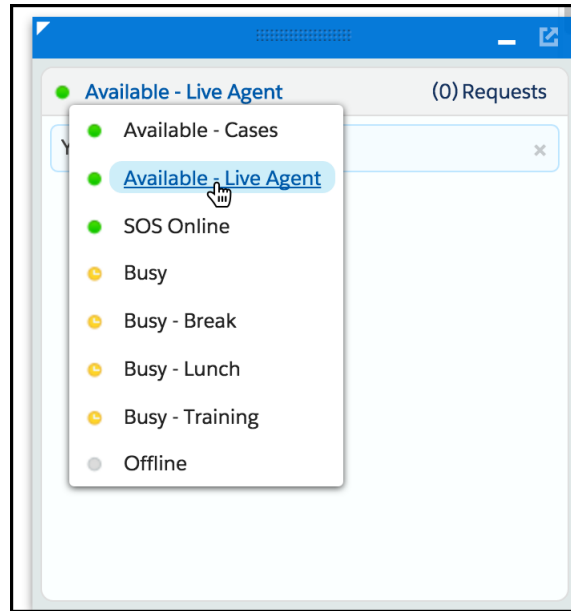
```
<a id="liveagent_button_online_575C00000004h3m"
  href="javascript://Chat"
  style="display: none;"
  onclick="liveagent.startChat('575C00000004h3m') ">
  <!-- Online Chat Content -->
</a>
<div id="liveagent_button_offline_575C00000004h3m"
  style="display: none;"
  <!-- Offline Chat Content -->
</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:

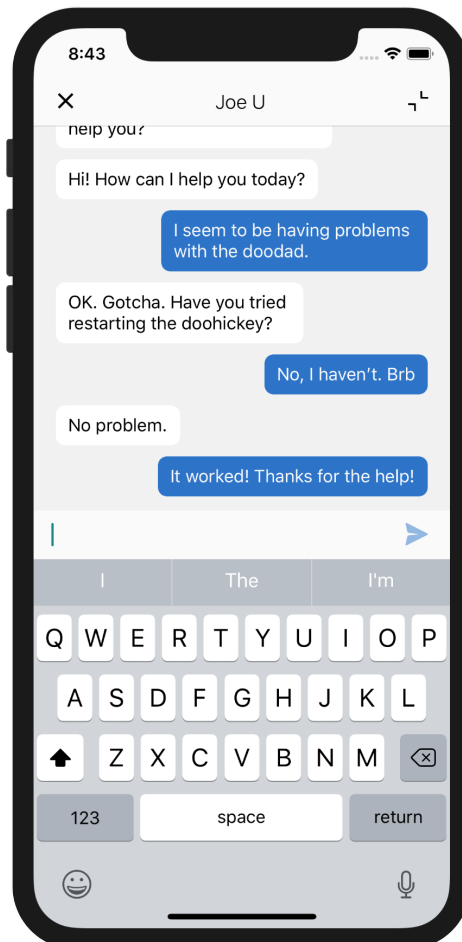
575C00000004h3m

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

7. Launch **Service Cloud Console**. From the **Omni-Channel** widget, ensure that an agent is online.



Now you can build and run the app. When you tap the **Chat** button, the app requests a chat session, which an agent can accept from the **Service Cloud Console**. From the console, an agent can real-time chat with a customer.



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 iOS](#)

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 iOS app, create an `SCSChatConfiguration` object and pass it to the `showChat` method.

[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.

[Notifications for Chat Activity](#)

If there's chat activity when the user is not viewing the chat session, you can present that information to them using the iOS notification system.

[Configure a Chat Session](#)

Before starting a chat session, you have several ways to configure the session using the `SCSChatConfiguration` object. These configuration settings allow you to specify pre-chat fields, determine whether a session starts minimized or full screen, and get updates about the user's queue position.

[Listen for State Changes and Events](#)

Implement `SCSChatSessionDelegate` to be notified about state changes made before, during, and after a chat session. This delegate also allows you to listen for error conditions so you can present alerts to the user when applicable.

[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.

[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](#)

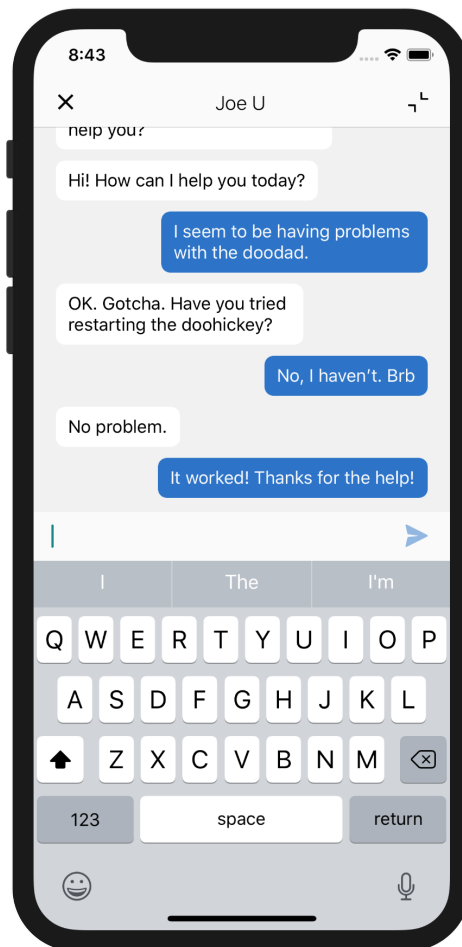
With the Chat Core API, you can access the functionality of Chat without a UI. This API is useful if you want to build your own UI and not use the default.

Chat in the Service Chat SDK for iOS

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

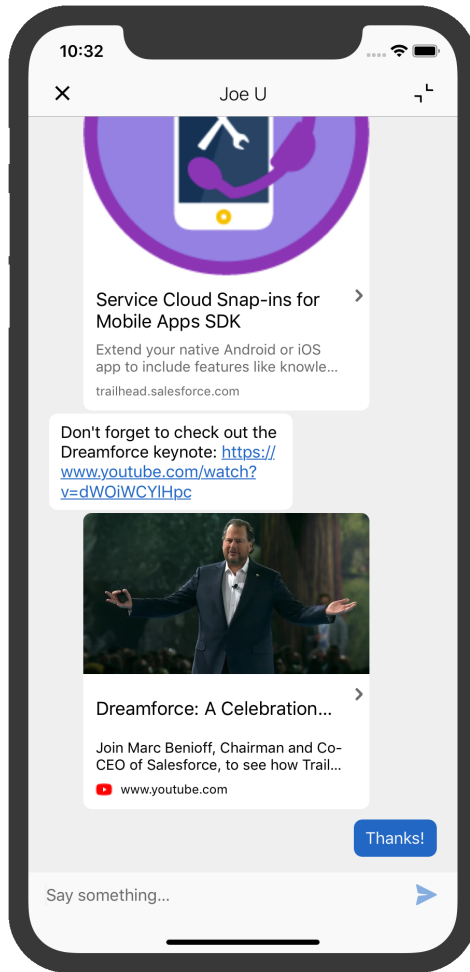
Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging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



This chat session can be minimized so that the user can continue to navigate from within the app while speaking with an agent. And if you've implemented Einstein Bots, the user first [speaks with a chat bot](#) on page 37 before optionally getting transferred to an agent.

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. These customizations include the ability to fine-tune the colors, the fonts, the images, and the strings used throughout the interface.

Quick Setup: Chat in the Service Chat SDK

To add Chat to your iOS app, create an `SCSChatConfiguration` object and pass it to the `showChat` method.

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 running through these steps, be sure you've already:

- Set up Service Cloud to work with Chat. To learn more, see [Org Setup for Chat in Lightning Experience with a Guided Flow](#).
- Installed the SDK. To learn more, see [Install the Service Chat SDK for iOS](#).

Once you've reviewed these prerequisites, you're ready to begin.

These steps describe how to set up Chat with the default UI. If you prefer to build your own user interface, see [Build Your Own UI with the Chat Core API](#).

1. Import the SDK. Wherever you intend to use the Chat SDK, be sure to import the ServiceCore framework and the ServiceChat framework.

In Swift:

```
import ServiceCore
import ServiceChat
```

In Objective-C:

```
@import ServiceCore;
@import ServiceChat;
```

2. Create an `SCSChatConfiguration` instance with information about your LiveAgent pod, your Salesforce org ID, the deployment ID, and the button ID.

In Swift:

```
let config = SCSChatConfiguration(liveAgentPod: "TO_DO_POD_NAME",
                                  // e.g. "d.gla5.gus.salesforce.com"
                                  orgId: "TO_DO_ORG_ID",
                                  // e.g. "00DB00000003Rxz"
                                  deploymentId: "TO_DO_DEPLOYMENT_ID",
                                  // e.g. "573B00000005KXz"
                                  buttonId: "TO_DO_BUTTON_ID")
                                  // e.g. "575C00000004h3m"
```

In Objective-C:

```
SCSChatConfiguration *config =
    [[SCSChatConfiguration alloc] initWithLiveAgentPod:@"TO_DO_POD_NAME"
                                                    // e.g. "d.gla5.gus.salesforce.com"
                                                    orgId:@"TO_DO_ORG_ID"
                                                    // e.g. "00DB00000003Rxz"
                                                    deploymentId:@"TO_DO_DEPLOYMENT_ID"
                                                    // e.g. "573B00000005KXz"
                                                    buttonId:@"TO_DO_BUTTON_ID"];
                                                    // e.g. "575C00000004h3m"
```



Note: You can get the required parameters for this method from your Salesforce org. See [Get Chat Settings from Your Org](#). If your Salesforce admin hasn't already set up Chat in Service Cloud or you need more guidance, see [Org Setup for Chat in Lightning Experience with a Guided Flow](#).

3. (Optional) Configure the visitor name, queue display style, whether the user can minimize the chat session, and various other configuration settings.

See [Configure a Chat Session](#) for more information.

4. (Optional) Customize the appearance with the configuration object.

You can configure the colors, fonts, and images to your interface with an `SCAppearanceConfiguration` instance. It contains the methods `setColor`, `setFontDescriptor`, and `setImage`. You can also configure the strings used throughout the interface. See [SDK Customizations with the Service Chat SDK for iOS](#).

5. (Optional) Specify any pre-chat fields.

You can specify both optional and required fields shown to the user before a chat session starts. You can also directly pass data to an agent without requiring any user input. These fields can be mapped directly to fields in a record in your org.

See [Show Pre-Chat Fields to User](#) and [Create or Update Salesforce Records from a Chat Session](#) for more information.


6. To start a chat session, call the `showChat(with:showPrechat:)` method on `SCSChatInterface`.

In Swift:

```
ServiceCloud.shared().chatUI.showChat(with: config!)
```

In Objective-C:


```
[[SCServiceCloud sharedInstance].chatUI showChatWithConfiguration:config];
```

 **Note:** The `showChat` method must be called on the main UI thread.


You can provide an optional completion block to execute code when the session has been fully connected to all services. During a successful session initialization, the SDK calls the completion block at the point that the session is active and the user is waiting for an agent to join. If there is a failure, the SDK calls the completion block with the associated error.

7. Listen for events and handle error conditions.

You can detect when a session ends by implementing the `session(didEnd:)` method on the `SCSChatSessionDelegate` delegate. Register this delegate using the `add(delegate:)` method on your `SCSChat` instance. In particular, we suggest that you handle the `.agent` reason (for when an agent ends a session) and the `.noAgentsAvailable` reason (for when there are no agents available). See [Listen for State Changes and Events](#).

 **Note:** The SDK doesn't show an alert when a session fails to start, or when a session ends. It's your responsibility to listen to events and display an error when appropriate.

These steps embed the chat experience into your app.

 **Note:** To learn about chat timeout limitations on iOS devices, see [When does a chat session time out?](#)

 **Example:** To use this example code, create a Single View Application and [Install the Service Chat SDK for iOS](#).

Use the storyboard to add a button to the view. Add a `Touch Up Inside` action in your `UIViewController` implementation with the name `startChat`. In the view controller code:

- Implement the `SCSChatSessionDelegate` protocol so that you can be notified when there are errors or state changes.
- Specify `self` as a chat delegate.
- Start a chat session in the button action.
- Implement the `session(didEnd:)` method and show a dialog when appropriate.

In Swift:

```
import UIKit
import ServiceCore
import ServiceChat

class ViewController : UIViewController, SCSChatSessionDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Add our chat delegate
        ServiceCloud.shared().chatCore.add(delegate: self)
    }
}
```

```

@IBAction func startChat(_ sender: AnyObject) {

    // Create config object
    if let config = SCSSChatConfiguration(liveAgentPod: "YOUR-POD-NAME",
                                          orgId: "YOUR-ORG-ID",
                                          deploymentId: "YOUR-DEPLOYMENT-ID",
                                          buttonId: "YOUR-BUTTON-ID") {

        // Start a session
        ServiceCloud.shared().chatUI.showChat(with: config)
    }
}

func session(_ session: SCSSChatSession!, didEnd endEvent: SCSSChatSessionEndEvent!)
{

    var description = ""

    // Here we'll handle the situation where the agent ends the session
    // and when there are no agents available...
    switch endEvent.reason {
    case .agent:
        description = "The agent has ended the session."
    case .noAgentsAvailable:
        description = "It looks like there are no agents available. Try again later."
    // TO DO: Handle other reasons
    default:
        description = "Session ended for an unknown reason."
    }

    let alert = UIAlertController(title: "Session Ended",
                                message: description,
                                preferredStyle: .alert)
    let okAction = UIAlertAction(title: "OK",
                                style: .default,
                                handler: nil)
    alert.addAction(okAction)
    self.present(alert, animated: true, completion: nil)
}

func session(_ session: SCSSChatSession!, didError error: Error!, fatal: Bool) {
    // TO DO: Handle error condition
    NSLog("Chat error: \(error.localizedDescription)")
}
}

```

In Objective-C:

```

#import "ViewController.h"
#import ServiceCore;
#import ServiceChat;

@interface ViewController : UIViewController <SCSSChatSessionDelegate>

@end

```

```

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Add our chat delegate
    [[SCServiceCloud sharedInstance].chatCore addDelegate:self];
}

- (IBAction)startChat:(id)sender {

    // Create config object
    SCSCChatConfiguration *config =
    [[SCSCChatConfiguration alloc] initWithLiveAgentPod:@"YOUR-POD-NAME"
                                                    orgId:@"YOUR-ORG-ID"
                                                    deploymentId:@"YOUR-DEPLOYMENT-ID"
                                                    buttonId:@"YOUR-BUTTON-ID"];

    // Start the session
    [[SCServiceCloud sharedInstance].chatUI showChatWithConfiguration:config];
}

- (void)session:(id<SCSCChatSession>)session didEnd:(SCSCChatSessionEndEvent *)endEvent
{

    NSString *description = nil;

    // Here we'll handle the situation where the agent ends the session
    // and when there are no agents available...
    switch (endEvent.reason){
        case SCSCChatEndReasonAgent:
            description = @"The agent has ended the session.";
            break;
        case SCSCChatEndReasonNoAgentsAvailable:
            description = @"It looks like there are no agents available. Try again later.";

            break;
        // TO DO: Handle other reasons
        default:
            description = @"Session ended for an unknown reason.";
            break;
    }

    UIAlertController *alert = [UIAlertController
                                alertControllerWithTitle:@"Session Ended"
                                message:description
                                preferredStyle:UIAlertControllerStyleAlert];

    UIAlertAction* okAction = [UIAlertAction
                                actionWithTitle:@"OK"
                                style:UIAlertActionStyleDefault
                                handler:nil];

    [alert addAction:okAction];
}

```



```
[self presentViewController:alert animated:YES completion:nil];
}

- (void)session:(id<SCSChatSession>)session didError:(NSError *)error fatal:(BOOL)fatal
{
    // TO DO: Handle error condition
    NSLog(@"Chat error: \(error.localizedDescription)");
}

@end
```

Use Einstein Bots with Chat

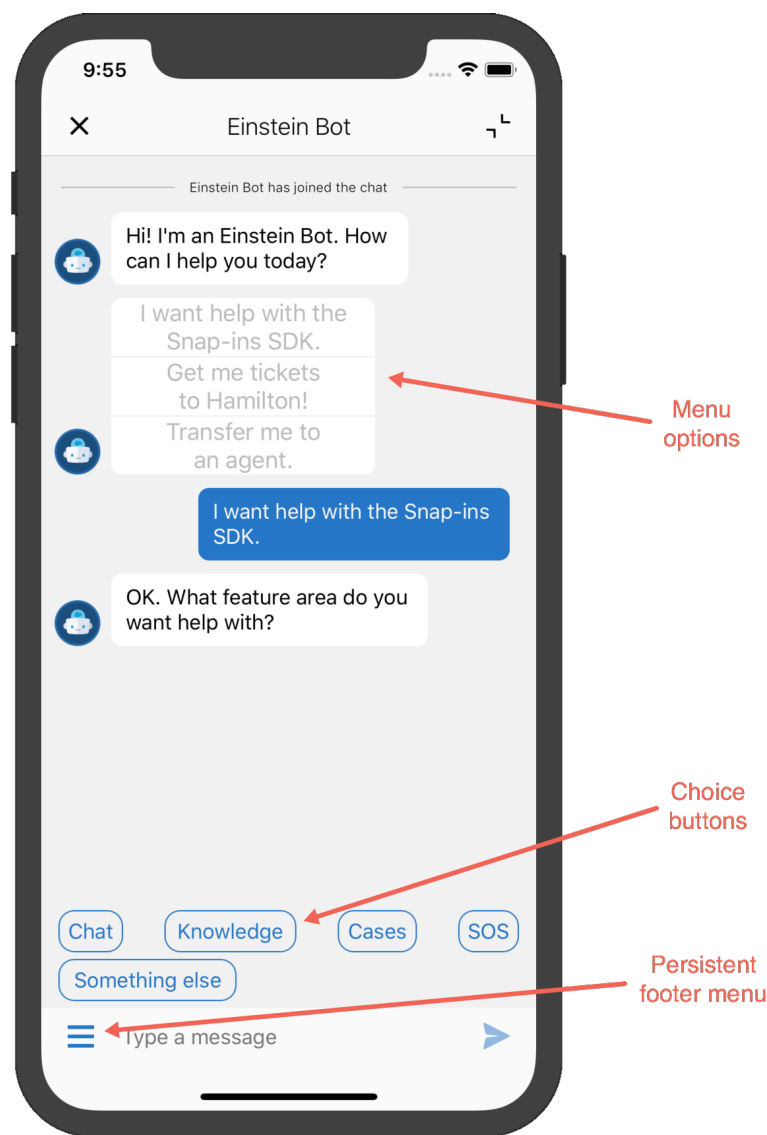
With Einstein Bots, you can complement your chat support experience with a smart, automated system that saves your agents time and keeps your customers happy. Once you've set up Einstein Bots in your org, the SDK automatically begins the chat experience using your bot. You can design your bot to transfer 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	Configure the bot avatar that displays during a session with a bot. To do this, use the <code>setImage</code> method with the <code>chatBotAvatar</code> enum value. To learn more, see Customize Images with the Service Chat SDK .
Einstein Bot Footer Icon	Configure the "hamburger" icon in the text entry area that launches the persistent footer menu, which is always accessible to the user. To do this, use the <code>setImage</code> method with the <code>chatBotFooterMenu</code> enum value. To learn more, see Customize Images with the Service Chat SDK .

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 `SCSAppEventList` 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 `SCSAppEventList` 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
let eventList = SCSAppEventList(scheme: "servicesdk")

// Add a regular expression for a path
eventList.addDescription("Tap for regex action", withExpression: "action\\a.*")

// Add a custom path for a specific path
eventList.addDescription("Tap for path action", forPath: "action/settings")
```

2. Add the event list to your `SCSChatConfiguration` object.

```
// Create a chat config object
let config = SCSChatConfiguration(liveAgentPod: "YOUR_POD_NAME",
                                orgId: "YOUR_ORG_ID",
                                deploymentId: "YOUR_DEPLOYMENT_ID",
                                buttonId: "YOUR_BUTTON_ID")

// ... Perform other chat configuration setup ...

// Add event list to chat configuration
config.eventList = eventList
```

3. Implement the `SCSChatInterfaceDelegate` delegate so that you can handle when the user taps on the URL.

```
class MyChatDelegate: SCSChatInterfaceDelegate {

    func interface(_ interface: SCSChatInterface!,
                  didReceiveAppEventWith URL: URL!)

        // Get path
        guard let fullPath = (URL as NSURL).resourceSpecifier else { return }

        // Minimize chat window
        ServiceCloud.shared().chatUI.minimize()

        if fullPath.contains("action/settings") {
```

```
        // TO DO: Perform some action for this url
        return
    }

    if fullPath.contains("action/alert") {
        // TO DO: Perform some action for this url
        return
    }

    // NOTE: You can maximize the chat window using
    //        ServiceCloud.shared().chatUI.maximize()
}
}
```

4. Add your delegate to the chat interface.

```
ServiceCloud.shared().chatUI.delegate = myChatDelegate
```

Notifications for Chat Activity

If there's chat activity when the user is not viewing the chat session, you can present that information to them using the iOS notification 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](#).

Notifications can be sent to the user when the user isn't viewing the chat session. A notification can appear when the app is in the background, or when the app is in the foreground but the chat session is minimized. 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
- Session will timeout soon

To ensure that the app can send these notifications while in the background, chat must be configured to allow background execution (`allowBackgroundExecution`) and to allow for background notifications (`allowBackgroundNotifications`). Both these settings are turned on by default. See [Configure a Chat Session](#) for details.

1. Import the `UserNotifications` framework in your `AppDelegate` class.

In Swift:

```
import UserNotifications
```

In Objective-C:

```
@import UserNotifications;
```

2. Register for local notifications in your `AppDelegate`'s `didFinishLaunchingWithOptions` method.

In Swift:

```
// Get the notification center object
let center = UNUserNotificationCenter.current()

// Register a delegate (see next step for delegate implementation)
center.delegate = self

// Request authorization
center.requestAuthorization(options: [.alert,.sound],
                             completionHandler: { granted, error in
    // Enable or disable features based on authorization
})

// Create general category
let generalCategory = UNNotificationCategory(identifier: "General", actions: [],
intentIdentifiers: [], options: .customDismissAction)
let categorySet: Set<UNNotificationCategory> = [generalCategory]

// Set category
center.setNotificationCategories(categorySet)
```

In Objective-C:

```
// Get the notification center object
UNUserNotificationCenter* center = [UNUserNotificationCenter currentNotificationCenter];

// Register a delegate (see next step for delegate implementation)
[center setDelegate:self];

// Request authorization
[center requestAuthorizationWithOptions:(UNAuthorizationOptionAlert +
UNAuthorizationOptionSound)
    completionHandler:^(BOOL granted, NSError * _Nullable error) {
    // Enable or disable features based on authorization
}];

// Create general category
UNNotificationCategory* generalCategory =
    [UNNotificationCategory
    categoryWithIdentifier:@"GENERAL"
        actions:@[]
        intentIdentifiers:@[]
        options:UNNotificationCategoryOptionCustomDismissAction];

// Set category
[center setNotificationCategories:[NSSet setWithObjects:generalCategory, nil]];
```

3. Implement `UNUserNotificationCenterDelegate`. Handle the `didReceiveNotificationResponse` and `willPresentNotification` methods.

iOS calls the `didReceiveNotificationResponse` method when your app is in the background. In this method, you can tell your app to enter the foreground and for chat to maximize. To perform this behavior, use the `handle(notification:)` method on the `SCSChatInterface` object, which is accessible from the `chatUI` property of the `ServiceCloud` shared instance.

iOS calls the `willPresentNotification` method when your app is in the foreground. To determine whether to display the notification, use the `shouldDisplayNotificationInForeground` method (accessible from the `chatUI` property of the `ServiceCloud` shared instance). If the Chat UI is already showing the relevant information related to this event, the method returns `false`.

In Swift:

```
/**
 This delegate method is executed when the application launches as a result
 of the user interacting with a notification when it is in the background.
 The result of passing the notification to Chat is that we will
 maximize if the notification was scheduled as a result of a chat event.
 */
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           didReceive response: UNNotificationResponse,
                           withCompletionHandler completionHandler:
                               @escaping () -> Void) {

    let chat = ServiceCloud.shared().chatUI!
    chat.handle(response.notification)
}

/**
 This delegate method is executed when a notification is received while
 the app is in the foreground. Check with Chat to see whether it's
 appropriate to display the notification (or if the notification
 relates to information already being shown to the user).
 */
func userNotificationCenter(_ center: UNUserNotificationCenter,
                           willPresent notification: UNNotification,
                           withCompletionHandler completionHandler:
                               @escaping (UNNotificationPresentationOptions) -> Void) {

    let chat = ServiceCloud.shared().chatUI!

    // Show we display this notification?
    if (chat.shouldDisplayNotificationInForeground()) {

        // Display notification as an alert
        completionHandler(.alert)
    }
}
```

In Objective-C:

```
/**
 This delegate method is executed when the application launches as a result
 of the user interacting with a notification when it is in the background.
 The result of passing the notification to Chat is that we will
 maximize if the notification was scheduled as a result of a chat event.
 */
- (void)userNotificationCenter:(UNUserNotificationCenter *)center
didReceiveNotificationResponse:(UNNotificationResponse *)response
withCompletionHandler:(void (^)(void))completionHandler {
```

```

SCSChatInterface *chat = [SCServiceCloud sharedInstance].chatUI;
[chat handleNotification:response.notification];
}

/**
This delegate method is executed when a notification is received while
the app is in the foreground. Check with Chat to see whether it's
appropriate to display the notification (or if the notification
relates to information already being shown to the user).
*/
- (void)userNotificationCenter:(UNUserNotificationCenter *)center
willPresentNotification:(UNNotification *)notification
withCompletionHandler:
(void (^)(UNNotificationPresentationOptions options))completionHandler {

    SCSChatInterface *chat = [SCServiceCloud sharedInstance].chatUI;


    // Show we display this notification?
    if ([chat shouldDisplayNotificationInForeground]) {

        // Display notification as an alert
        completionHandler(UNNotificationPresentationOptionAlert);
    }
}

```

Configure a Chat Session

Before starting a chat session, you have several ways to configure the session using the `SCSChatConfiguration` object. These configuration settings allow you to specify pre-chat fields, determine whether a session starts minimized or full screen, and get updates about the user's queue position.

 **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 you start a chat session, you specify an `SCSChatConfiguration` object as one of the arguments. This object contains all the configuration settings necessary for Chat to start a session. To create an `SCSChatConfiguration` object, you specify information about your org and deployment.

In Swift:


```

let config = SCSChatConfiguration(liveAgentPod: "TO_DO_POD_NAME",
                                  // e.g. "d.gla5.gus.salesforce.com"
                                  orgId: "TO_DO_ORG_ID",
                                  // e.g. "00DB00000003Rxx"
                                  deploymentId: "TO_DO_DEPLOYMENT_ID",
                                  // e.g. "573B00000005KXz"
                                  buttonId: "TO_DO_BUTTON_ID")
                                  // e.g. "575C00000004h3m"

```

In Objective-C:

```
SCSChatConfiguration *config =
    [[SCSChatConfiguration alloc] initWithLiveAgentPod:@"TO_DO_POD_NAME"
                                                    // e.g. "d.gla5.gus.salesforce.com"
                                                    orgId:@"TO_DO_ORG_ID"
                                                    // e.g. "00DB00000003Rxxz"
                                                    deploymentId:@"TO_DO_DEPLOYMENT_ID"
                                                    // e.g. "573B00000005KXz"
                                                    buttonId:@"TO_DO_BUTTON_ID"];
                                                    // e.g. "575C00000004h3m"
```

 **Note:** You can get the required parameters for this method from your Salesforce org. See [Get Chat Settings from Your Org](#). If your Salesforce admin hasn't already set up Chat in Service Cloud or you need more guidance, see [Org Setup for Chat in Lightning Experience with a Guided Flow](#).

However, there are other options you can set using `SCSChatConfiguration` at configuration time.

The following features are available for configuration:

Table 1: Session Display Configuration Settings

Property Name	Description	Type & Default Value
<code>allowMinimization</code>	Indicates whether the user is allowed to minimize the chat session view.	Bool: true/YES
<code>allowURLPreview</code>	Indicates whether the user is shown URL previews when the agent types a URL in the chat feed.	Bool: true/YES
<code>defaultToMinimized</code>	Indicates whether the chat session starts out as a minimized thumbnail view.	Bool: true/YES


 **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.

Table 2: Background Configuration Settings

Property Name	Description	Type & Default Value
<code>allowBackgroundExecution</code>	Indicates whether to allow extended background execution to support active chat sessions. When <code>true</code> , active chat sessions can remain in the background for more than three minutes. See <code>allowBackgroundNotifications</code> for related functionality.	Bool: true/YES
<code>allowBackgroundNotifications</code>	Indicates whether the session posts local notifications based on chat activity. Requires that <code>allowBackgroundExecution</code> is also set to <code>true</code> . To learn more, see Notifications for Chat Activity .	Bool: true/YES


 **Note:** When turning on background execution, be sure that `Idle Connection Timeout Duration` is set in your org. To learn more, see [Chat Deployment Settings](#).

Table 3: Pre-Chat Configuration Settings

Property Name	Description	Type & Default Value
<code>prechatEntities</code>	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 an <code>SCSPrechatEntity</code> for each Salesforce object (for example, Case or Contact) and instantiate an <code>SCSPrechatEntityField</code> for each field association within that Salesforce object (for example, Subject or LastName). To learn more, see Create or Update Salesforce Records from a Chat Session .	<code>SCSPrechatEntity</code> array: nil
<code>prechatFields</code>	You can specify both optional and required fields shown to the user before a chat session starts. You can also directly pass data to an agent without requiring any user input. To create pre-chat fields, add <code>SCSPrechatObject</code> instances to the <code>prechatFields</code> property on the <code>SCSChatConfiguration</code> object. To learn more, see Show Pre-Chat Fields to User .	<code>SCSPrechatObject</code> array: nil

Table 4: Other Configuration Settings

Property Name	Description	Type & Default Value
<code>eventList</code>	Adds a custom path that displays a custom tile and performs a custom action within your app. To learn more, see Handle Custom URLs in Chat on page 39.	<code>SCSAppEventList</code>
<code>queueStyle</code>	Determines the style of the queue. The queue can either display the queue position number (<code>.position</code>), the estimated wait time (<code>.estimatedWaitTime</code>), or no queue information at all (<code>.none</code>). You can subscribe to queue position events using <code>session(didUpdateQueuePosition:estimatedWaitTime:)</code> on <code>SCSChatSessionDelegate</code> . Use the <code>add(delegate:)</code> method on <code>SCSChat</code> to register your delegate. When using the estimated wait time, the wait time is shown to the user in minutes. However, you can set the minimum (<code>minimumEstimatedWaitTime</code>) and maximum (<code>maximumEstimatedWaitTime</code>) wait time values. If the wait time exceeds the maximum value, a generic message appears, which you can customize (using the customizable chat strings in the <code>ServiceCloud.Chat.Status.EstimatedWait</code> namespace). To understand the algorithm used for the estimated wait time, see the estimated wait time documentation in the Chat REST API Developer Guide .	<code>SCSChatConfiguration</code> <code>QueueStyle: .position</code>

Property Name	Description	Type & Default Value
	<p>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.</p> $q = \max(n - c, 0)$ <p>Where:</p> <ul style="list-style-type: none"> • 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 <p>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.</p>	
remoteLoggingEnabled	Indicates whether session logs are sent for collection. (Logs sent remotely don't collect personal information. Unique IDs are created for tying logs to sessions and those IDs can't be correlated back to specific users.)	Bool: true/YES
visitorName	Name of the chat visitor. This value is used by the Service Cloud console and displayed to the agent.	String: "Visitor"

Once you've fully configured the `SCSChatConfiguration` object, you can start the session using the `startSession` method.



Example: The following example configures a session with one pre-chat field and a visitor name "Jane Doe".

In Swift:

```
let config = SCSChatConfiguration(liveAgentPod: "YOUR_POD_NAME",
                                orgId: "YOUR_ORG_ID",
                                deploymentId: "YOUR_DEPLOYMENT_ID",
                                buttonId: "YOUR_BUTTON_ID")

// Set the visitor name
config?.visitorName = "Jane Doe"

// Change from queue position to estimated wait time
config?.queueStyle = .estimatedWaitTime

// Add a required email field (with an email keyboard and no auto-correction)
let emailField = SCSPrechatTextInputObject(label: "Email")
emailField?.isRequired = true
emailField?.keyboardType = .emailAddress
emailField?.autocorrectionType = .no
config?.prechatFields.append(emailField!)
```

In Objective-C:

```
SCSChatConfiguration *config =
    [[SCSChatConfiguration alloc] initWithLiveAgentPod:@"YOUR_POD_NAME"
                                                orgId:@"YOUR_ORG_ID"
                                                deploymentId:@"YOUR_DEPLOYMENT_ID"
                                                buttonId:@"YOUR_BUTTON_ID"];

// Set the visitor name
config.visitorName = @"Jane Doe";

// Change from queue position to estimated wait time
config.queueStyle = SCSChatConfigurationQueueStyleEstimatedWaitTime;

// Add a required email field (with an email keyboard and no auto-correction)
SCSPrechatTextInputObject* emailField = [[SCSPrechatTextInputObject alloc]
                                           initWithLabel:@"Email"];
emailField.required = YES;
emailField.keyboardType = UIKeyboardTypeEmailAddress;
emailField.autocorrectionType = UITextAutocorrectionTypeNo;
[config.prechatFields addObject:emailField];
```

Listen for State Changes and Events

Implement `SCSChatSessionDelegate` to be notified about state changes made before, during, and after a chat session. This delegate also allows you to listen for error conditions so you can present alerts to the user when applicable.



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](#).

Listening to State Changes

A chat session can be in one of the following states:

Inactive

No active session.

Connecting

A connection with chat servers is being established.

Queued

A connection has been established, and is now in the queue for next available agent.

Connected

Connected with an agent.

Ending

Session is cleaning up the connection at the end of a session.

These states are defined in `SCSChatSessionState`.

Throughout a session, your application might want to know the current state. You can monitor state changes by implementing `SCSChatSessionDelegate`. Use the `add(delegate:)` method on `SCSChat` to register your delegate. Use the `session(didTransitionFrom:to:)` method to listen for state changes.

Handling Session Termination and Error Conditions

The SDK doesn't present UI alerts for session termination or error conditions so you'll need to listen for these events and decide what to show your users. There are two `SCSChatSessionDelegate` methods for this purpose:

1. To track session termination, use the `session(didEnd:)` method. Inspect the reason (`SCSChatSessionEndEvent.type`) to determine why the session stopped. Use the `SCSChatSession` object to get the session ID and other session information.
2. You can track errors with the `session(didError:fatal:)` method. Compare the error code to `SCSChatErrorCode` to determine what kind of error occurred.



Example: This sample code does the following:

- Implements the `SCSChatSessionDelegate` protocol.
- Implements the `session(didTransitionFrom:to:)` method to listen for state changes.
- Implements the `session(didEnd:)` method and logs a few possible end reasons.
- Implements the `session(didError:fatal:)` method to listen for errors.

```
import UIKit
import ServiceCore
import ServiceChat

class MyChatSessionDelegateImplementation: NSObject, SCSChatSessionDelegate {

    // TO DO: Register this delegate using
    //      ServiceCloud.shared().chatCore.add(delegate: myDelegate)

    /**
     Delegate method to handle state change.
     */
    func session(_ session: SCSChatSession!,
                 didTransitionFrom previous: SCSChatSessionState,
                 to current: SCSChatSessionState) {

        NSLog("Chat state changed...")

        switch current {
        case .connecting:
            NSLog("Chat now connecting...")
        case .connected:
            NSLog("Chat connected...")
            // TO DO: Handle other reasons
        default:
            break
        }
    }

    /**
     Delegate method for session stop event.
     */
}
```

```

func session(_ session: SCSCChatSession!, didEnd endEvent: SCSCChatSessionEndEvent!)
{
    var reason = "Unknown"

    switch endEvent.reason {
    case .agent:
        reason = "The agent has ended the session."
    case .noAgentsAvailable:
        reason = "No agents were available."
    default:
        // TO DO: Handle other reasons
        break
    }

    NSLog("\nChat End Session. Reason: \(reason)")


    // You can access the session ID from the SCSCChatSession object
    let sessionId = session.sessionId
}

/**
Delegate method for error conditions.
*/
func session(_ session: SCSCChatSession!, didError error: Error!, fatal: Bool) {
    // TO DO: Handle error condition
    NSLog("Chat error: \(error.localizedDescription)")
}
}

```

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, add `SCSPrechatObject` instances to the `prechatFields` property on the `SCSCChatConfiguration` object.

1. Create an `SCSCChatConfiguration` object.

In Swift:

```

let config = SCSCChatConfiguration(liveAgentPod: "TO_DO_POD_NAME",
                                   // e.g. "d.gla5.gus.salesforce.com"
                                   orgId: "TO_DO_ORG_ID",
                                   // e.g. "00DB00000003Rxx"
                                   deploymentId: "TO_DO_DEPLOYMENT_ID",
                                   // e.g. "573B00000005KXz"

```

```
buttonId: "TO_DO_BUTTON_ID")
// e.g. "575C00000004h3m"
```

In Objective-C:

```
SCSChatConfiguration *config =
    [[SCSChatConfiguration alloc] initWithLiveAgentPod:@"TO_DO_POD_NAME"
                                                // e.g. "d.gla5.gus.salesforce.com"
                                                orgId:@"TO_DO_ORG_ID"
                                                // e.g. "00DB00000003Rxx"
                                                deploymentId:@"TO_DO_DEPLOYMENT_ID"
                                                // e.g. "573B00000005KXz"
                                                buttonId:@"TO_DO_BUTTON_ID"];
// e.g. "575C00000004h3m"
```

See [Configure a Chat Session](#) on how to configure a chat session.

2. Create [SCSPrechatObject](#) objects for the pre-chat fields you want to specify in your app.

There are several types of pre-chat fields:

- [SCSPrechatObject](#) does not require user input and can be used to send custom data directly to the agent.
- [SCSPrechatTextInputObject](#) (a subclass of [SCSPrechatObject](#)) takes user input from a text field.
- [SCSPrechatPickerObject](#) (a subclass of [SCSPrechatObject](#)) provides the user with a dropdown list of options.

Each type has different properties you can configure.

- a. Create objects that don't require user input using [SCSPrechatObject](#).

In Swift:

```
let customData = SCSPrechatObject(label: "CustomEmailField",
                                   value: "lauren@example.com")
```

In Objective-C:

```
SCSPrechatObject* customData = [[SCSPrechatObject alloc]
                                initWithLabel:@"CustomEmailField"
                                value:@"lauren@example.com"];
```

When using [SCSPrechatObject](#) to send data without user input, specify both the label and the value. The [SCSPrechatObject](#) base class contains the following properties:

- `label`—name of the pre-chat field shown to agent.
- `value`—value of the pre-chat field; only use this property if you don't intend for the user to fill in this field.
- `displayLabel`—optional display name of the pre-chat field shown to the user if different than the label.
- `transcriptFields`—optional array of field identifiers on the `LiveAgentChatTranscript` object in Salesforce.
- `displayToAgent`—indicates whether this pre-chat detail is shown to an agent accepting the chat session; defaults to `true`.

- b. Create text input objects using [SCSPrechatTextInputObject](#).

In Swift:

```
// Create a text field
let myPrechatField = SCSPrechatTextInputObject(label: "Full Name")
```

In Objective-C:

```
// Create a text field
SCSPrechatTextInputObject* myPrechatField = [[SCSPrechatTextInputObject alloc]
initWithLabel:@"Full Name"];
```

When using a [SCSPrechatTextInputObject](#), you can control several other properties:

- `required`—indicates whether the field is required.
- `keyboardType`—provides access to other standard keyboards (such as `UIKeyboardTypeEmailAddress`).
- `autocapitalizationType`—controls how text capitalization works.
- `autocorrectionType`—controls auto-correction behavior.
- `maxLength`—specifies the maximum length of the field
- `displayLabel`—optional display name of the pre-chat field shown to the user if different than the label.
- `transcriptFields`—optional array of field identifiers on the `LiveAgentChatTranscript` object in Salesforce.
- `displayToAgent`—indicates whether this pre-chat detail is shown to an agent accepting the chat session; defaults to `true`.

c. Create dropdown lists using [SCSPrechatPickerObject](#).

In Swift:

```
// Create dropdown choices
let statusOptions = NSMutableArray()
statusOptions.add(SCSPrechatPickerOption(label:"New Issue", value:"New"))
statusOptions.add(SCSPrechatPickerOption(label:"Fixed Issue", value:"Fixed"))

// Create a dropdown list with choices
let statusPickerField = SCSPrechatPickerObject(
    label: "Issue Type",
    options: statusOptions as! [SCSPrechatPickerOption])
```

In Objective-C:

```
// Create a dropdown list with two choices
NSMutableArray *statusOptions = [[NSMutableArray alloc] init];
[statusOptions addObject: [[SCSPrechatPickerOption alloc]
initWithLabel:@"New Issue" value:@"New"]];
[statusOptions addObject: [[SCSPrechatPickerOption alloc]
initWithLabel:@"Fixed Issue" value:@"Fixed"]];

// Create a dropdown list with choices
SCSPrechatPickerObject *picker = [[SCSPrechatPickerObject alloc]
initWithLabel:@"Issue Type" options:statusOptions];
```

When using a [SCSPrechatPickerObject](#), you can access these properties:

- `required`—indicates whether the field is required.
- `options`—specifies items in the dropdown list. This property is an array of [SCSPrechatPickerOption](#) objects.
- `displayLabel`—optional display name of the pre-chat field shown to the user if different than the label.
- `transcriptFields`—optional array of field identifiers on the `LiveAgentChatTranscript` object in Salesforce.
- `displayToAgent`—indicates whether this pre-chat detail is shown to an agent accepting the chat session; defaults to `true`.

3. (Optional) Create `SCSPrechatEntity` 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 an `SCSPrechatEntity` for each Salesforce object (for example, Case or Contact) and instantiate an `SCSPrechatEntityField` for each field association within that Salesforce object (for example, Subject or LastName). To learn more, see [Create or Update Salesforce Records from a Chat Session](#).

4. Update the config object's `prechatFields` property with an array of your pre-chat objects.

In Swift:

```
// Add the array of pre-chat fields to the config object
config?.prechatFields = myPrechatFields
```

In Objective-C:

```
// Add the array of pre-chat fields to the config object
config.prechatFields = myPrechatFields;
```

5. Show the pre-chat form and start a chat session by calling `showChat(with:showPrechat:)` and specify `true` for whether to show the pre-chat form.
 - a. If you want to show the pre-chat form and then send those results to the agent when starting a session, call `showChat(with:showPrechat:)`.

In Swift:

```
// Show the pre-chat form and start a session
ServiceCloud.shared().chatUI.showChat(with: config, showPrechat: true)
```

In Objective-C:

```
// Show the pre-chat form and start a session
[[SCServiceCloud sharedInstance].chatUI showChatWithConfiguration:config showPrechat:
YES];
```

- b. If you want to programmatically change the pre-chat data before passing it to the org or agent, call the `showPrechat(withFields:modal:completion:)` method first. From within the completion block of this method, update the config object's `prechatFields` property and then call `showChat(with:showPrechat:)` and specify `false` for `showPrechat`.

For example, the following Swift code asks the user for their first and last name. After the user completes the pre-chat form, the code creates a new field by concatenating the two name fields. When we start the chat session, only this new field is sent to the agent.

```
// Create two text fields
let firstNameField = SCSPrechatTextInputObject(label: "First Name")
let lastNameField = SCSPrechatTextInputObject(label: "Last Name")
let prechatInputFields = [firstNameField, lastNameField] as? [SCSPrechatObject]

// Show the pre-chat form...
ServiceCloud.shared().chatUI.showPrechat(withFields: prechatInputFields, completion:
{
    (prechatResultFields, completed) in

    // If the pre-chat form completed successfully...
    if (completed && prechatResultFields != nil && prechatResultFields!.count >= 2) {
```



```

    // Create the full name from the values of the two pre-chat name fields
    let fullName = prechatResultFields![0].value + " " + prechatResultFields![1].value

    // Create a full name field
    let fullNameField = SCSPrechatObject(label: "Full Name", value: fullName)

    // Add this new field to the config object
    // (We DON'T include the original pre-chat fields because we don't need to send
    them...)
    config!.prechatFields = [fullNameField]

    // And now start a chat session (without showing pre-chat)
    ServiceCloud.shared().chatUI.showChat(with: config!)

} else {
    // TO DO: Handle the scenario where the user cancels out of the pre-chat form
}
})

```



Example: This code sample builds a set of pre-chat fields that are shown to the user.

```

let config = SCSChatConfiguration(liveAgentPod: "YOUR-POD-NAME",
                                orgId: "YOUR-ORG-ID",
                                deploymentId: "YOUR-DEPLOYMENT-ID",
                                buttonId: "YOUR-BUTTON-ID")

// Add some required fields
let firstNameField = SCSPrechatTextInputObject(label: "First Name")
firstNameField!.isRequired = true
let lastNameField = SCSPrechatTextInputObject(label: "Last Name")
lastNameField!.isRequired = true
let emailField = SCSPrechatTextInputObject(label: "Email")
emailField!.isRequired = true
emailField!.keyboardType = .emailAddress
emailField!.autocorrectionType = .no

// Add some optional fields
let originField = SCSPrechatTextInputObject(label: "Where are you from?")
originField!.isRequired = false
let phoneField = SCSPrechatTextInputObject(label: "Phone Number")
phoneField!.isRequired = false
phoneField!.keyboardType = .phonePad
let descriptionField = SCSPrechatTextInputObject(label: "Please describe your problem:")
descriptionField!.isRequired = false

// Add a picklist field
let statusOptions = NSMutableArray()
statusOptions.add(SCSPrechatPickerOption(label:"New Issue", value:"New"))
statusOptions.add(SCSPrechatPickerOption(label:"Fixed Issue", value:"Fixed"))
let statusPickerField = SCSPrechatPickerObject(label: "Status",
options: statusOptions as! NSArray as! [SCSPrechatPickerOption])
statusPickerField!.isRequired = false

```

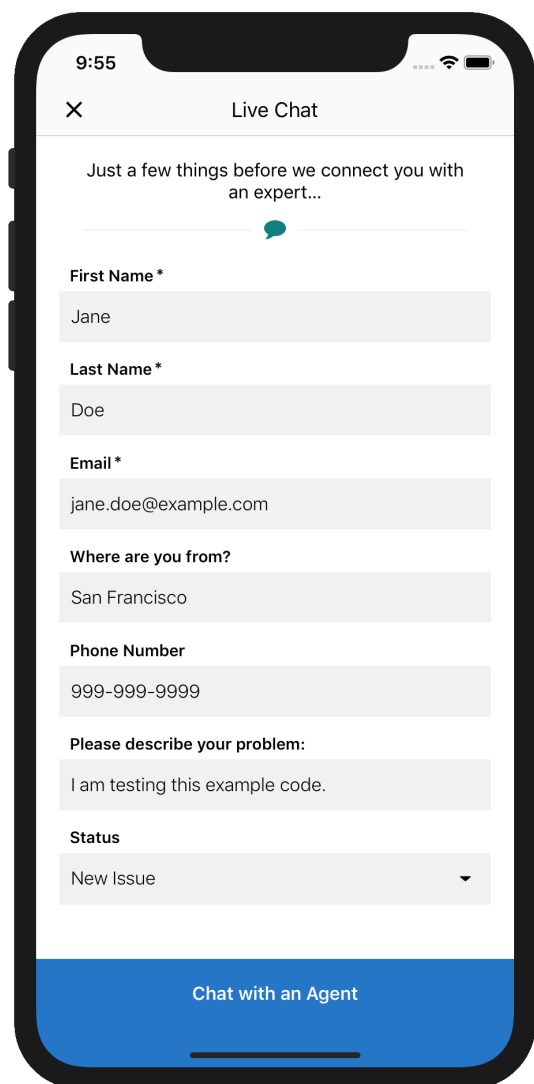
```
// Add hidden field containing the subject
let subjectField = SCSPrechatObject(label: "Subject", value: "Chat Session")

// Create an array of all pre-chat fields
let prechatFields: [SCSPrechatObject] = [firstNameField!, lastNameField!, emailField!,
                                          originField!, phoneField!,
descriptionField!,
                                          statusPickerField!, subjectField]

// Add the array of pre-chat fields to the config object
config?.prechatFields = prechatFields

// And NOW show the chat UI
ServiceCloud.shared().chatUI.showChat(with: config!, showPrechat: true)
```

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



9:55

✕ Live Chat

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

First Name *

Jane

Last Name *

Doe

Email *

jane.doe@example.com

Where are you from?

San Francisco

Phone Number

999-999-9999

Please describe your problem:

I am testing this example code.

Status

New Issue

Chat with an Agent

And the agent sees the following UI from the console.

The screenshot shows a 'Chat - Visitor' window with a sidebar containing an 'End Chat' button. The main area displays 'Live Agent Chat Visitor' and a 'Visitor Details' table.

IP Address	[Redacted]	Deployment Name	[Redacted]
Network	Salesforce.com	Location	San Francisco, CA, United States
Language	n/a	Original Referrer	
Chat Requested Time	[Redacted]	Current Page	
Browser	[Redacted]	Screen Resolution	n/a
Visitor		Email	jane.doe@example.com
First Name	Jane	Last Name	Doe
Phone Number	999-999-9999	Please describe your problem:	I am testing this example code.
Status	New	Subject	Live Agent Chat Session
Where are you from?	San Francisco		

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 an `SCSPrechatEntity` for each Salesforce object (for example, `Case` or `Contact`) and instantiate an `SCSPrechatEntityField` for each field association within that Salesforce object (for example, `Subject` or `LastName`).

Each pre-chat entity must map to a pre-chat object (`SCSPrechatObject`, `SCSPrechatTextInputObject`, or `SCSPrechatPickerObject`). The label string in this pre-chat object must be identical to the label used in your `SCSPrechatEntityField` object.

Use the config object's `prechatFields` property for the array of your pre-chat objects and the `prechatEntities` property for the array of your entity objects.

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

In Swift:

```
// Create pre-chat fields
let firstNameField = SCSPrechatObject(label: "First Name", value: "Jane")
let lastNameField = SCSPrechatObject(label: "Last Name", value: "Doe")

// Create entity fields
let firstNameEntityField =
    SCSPrechatEntityField(fieldName: "FirstName", label: "First Name")
firstNameEntityField.doFind = true
firstNameEntityField.doCreate = true
let lastNameEntityField =
    SCSPrechatEntityField(fieldName: "LastName", label: "Last Name")
lastNameEntityField.doFind = true
lastNameEntityField.doCreate = true

// Create an entity object
let contactEntity =
    SCSPrechatEntity(entityName: "Contact")
contactEntity.showOnCreate = true

// Add entity fields to entity object
contactEntity.entityFieldsMaps.add(firstNameEntityField)
contactEntity.entityFieldsMaps.add(lastNameEntityField)

// Update config object with the pre-chat fields
config!.prechatFields = [firstNameField, lastNameField]

// Update config object with the entity mappings
config!.prechatEntities = [contactEntity]
```

In Objective-C:

```
// Create pre-chat fields
SCSPrechatObject* firstNameField = [[SCSPrechatObject alloc]
                                     initWithLabel:@"First Name"
                                     value:@"Banana"];
SCSPrechatObject* lastNameField = [[SCSPrechatObject alloc]
                                    initWithLabel:@"Last Name"
                                    value:@"Town"];

// Create entity fields
SCSPrechatEntityField* firstNameEntityField =
    [[SCSPrechatEntityField alloc]
     initWithFieldName:@"FirstName" label:@"First
Name"];
firstNameEntityField.doFind = YES;
firstNameEntityField.doCreate = YES;
SCSPrechatEntityField* lastNameEntityField =
    [[SCSPrechatEntityField alloc]
     initWithFieldName:@"LastName" label:@"Last Name"];
lastNameEntityField.doFind = YES;
lastNameEntityField.doCreate = YES;

// Create an entity object
```

```
SCSPrechatEntity* contactEntity = [[SCSPrechatEntity alloc]
                                   initWithEntityName:@"Contact"];
contactEntity.showOnCreate = YES;

// Add entity fields to entity object
[contactEntity.entityFieldsMaps addObject:firstNameEntityField];
[contactEntity.entityFieldsMaps addObject:lastNameEntityField];

// Update config object with the pre-chat fields
NSMutableArray<SCSPrechatObject *> *preChatFields = [NSMutableArray new];
[preChatFields addObject:firstNameField];
[preChatFields addObject:lastNameField];
config.prechatFields = preChatFields;

// Update config object with the entity mappings
NSMutableArray<SCSPrechatEntity *> *prechatEntities = [NSMutableArray new];
[prechatEntities addObject:contactEntity];
config.prechatEntities = prechatEntities;
```

Entity Configuration Settings

The `SCSPrechatEntity` and `SCSPrechatEntityField` classes give you various configuration settings for mapping fields. For example, if a field doesn't exist, you can have the SDK create that field. The following code sample illustrates some basic building blocks when creating an `SCSPrechatEntity` object.

In Swift:

```
// Create an entity
let entity = SCSPrechatEntity(entityName: "Contact")
entity.saveToTranscript = "ContactId" // Save this entity to Transcript.ContactId
entity.linkToEntityName = "Case"
entity.linkToEntityField = "ContactId" // Link this entity to Case.ContactId

// Add an entity field map to our entity
let entityField = SCSPrechatEntityField(fieldName: "FirstName", label: "First Name")
entityField.doFind = true // Attempt to search for that field
entityField.isExactMatch = true // Must be an exact match
entityField.doCreate = true // Create if not found
entity.entityFieldsMaps.add(entityField) // Add field to entity map
```

In Objective-C:

```
// Create an entity
SCSPrechatEntity* entity = [[SCSPrechatEntity alloc] initWithEntityName:@"Contact"];
entity.saveToTranscript = @"ContactId"; // Save this entity to Transcript.ContactId
entity.linkToEntityName = @"Case";
entity.linkToEntityField = @"ContactId"; // Link this entity to Case.ContactId

// Add an entity field map to our entity
SCSPrechatEntityField* entityField = [[SCSPrechatEntityField alloc]
                                       initWithFieldName:@"FirstName" label:@"First Name"];
entityField.doFind = YES; // Attempt to search for that field
entityField.isExactMatch = YES; // Must be an exact match
entityField.doCreate = YES; // Create if not found
```

```
[entity.entityFieldsMaps
  addObject:entityField];          // Add field to entity map
```

See the reference documentation for [SCSPrechatEntity](#) and [SCSPrechatEntityField](#). Also refer to [Chat REST API Data Types](#) for the Entity and EntityFieldsMaps data types, which define the underlying functionality of these SDK objects.



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

```
let config = SCSCChatConfiguration(liveAgentPod: "YOUR_POD_NAME",
                                   orgId: "YOUR_ORG_ID",
                                   deploymentId: "YOUR_DEPLOYMENT_ID",
                                   buttonId: "YOUR_BUTTON_ID")

// Create some basic pre-chat fields (with user input)
let firstNameField = SCSPrechatTextInputObject(label: "First Name")
firstNameField!.isRequired = true
let lastNameField = SCSPrechatTextInputObject(label: "Last Name")
lastNameField!.isRequired = true
let emailField = SCSPrechatTextInputObject(label: "Email")
emailField!.isRequired = true
emailField!.keyboardType = .emailAddress
emailField!.autocorrectionType = .no

// Create a pre-chat field without user input
let subjectField = SCSPrechatObject(label: "Subject", value: "Chat Session")

// Create an entity mapping for a Contact record type
let contactEntity = SCSPrechatEntity(entityName: "Contact")
contactEntity.saveToTranscript = "Contact"
contactEntity.linkToEntityName = "Case"
contactEntity.linkToEntityField = "ContactId"

// Add some field mappings to our Contact entity
let firstNameEntityField = SCSPrechatEntityField(fieldName: "FirstName", label: "First
  Name")
firstNameEntityField.doFind = true
firstNameEntityField.isExactMatch = true
firstNameEntityField.doCreate = true
contactEntity.entityFieldsMaps.add(firstNameEntityField)
let lastNameEntityField = SCSPrechatEntityField(fieldName: "LastName", label: "Last
  Name")
lastNameEntityField.doFind = true
lastNameEntityField.isExactMatch = true
lastNameEntityField.doCreate = true
contactEntity.entityFieldsMaps.add(lastNameEntityField)
let emailEntityField = SCSPrechatEntityField(fieldName: "Email", label: "Email")
emailEntityField.doFind = true
emailEntityField.isExactMatch = true
emailEntityField.doCreate = true
contactEntity.entityFieldsMaps.add(emailEntityField)

// Create an entity mapping for a Case record type
let caseEntity = SCSPrechatEntity(entityName: "Case")
caseEntity.saveToTranscript = "Case"
```

```

caseEntity.showOnCreate = true

// Add one field mappings to our Case entity
let subjectEntityField = SCSPrechatEntityField(fieldName: "Subject", label: "Subject")
subjectEntityField.doCreate = true
caseEntity.entityFieldsMaps.add(subjectEntityField)

// Update config object with the pre-chat fields
config!.prechatFields =
    [firstNameField, lastNameField, emailField, subjectField] as? [SCSPrechatObject]

// Update config object with the entity mappings
config!.prechatEntities = [contactEntity, caseEntity]

// Start the session!
ServiceCloud.shared().chatUI.showChat(with: config!, showPrechat: true)

```

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.



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.

To check whether agents are available, call the [determineAvailabilityWithConfiguration](#) method on the `chatCore` property, similar to how you [start a chat session](#).

In Swift:

```

let config = SCSChatConfiguration(liveAgentPod: "YOUR-POD-NAME",
                                orgId: "YOUR-ORG-ID",
                                deploymentId: "YOUR-DEPLOYMENT-ID",
                                buttonId: "YOUR-BUTTON-ID")

ServiceCloud.shared().chatCore.determineAvailability(with: config,
                                                    completion: { (error: Error?,
                                                                    available: Bool,
                                                                    estimatedWaitTime: TimeInterval) in

    if (error != nil) {
        // TO DO: Handle error
    }
    else if (available) {
        // TO DO: Enable chat button...

        // Optionally, use the estimatedWaitTime to
        // show an estimated wait time until an agent
    }
}

```

```

        // is available. This value is only valid if
        // SCSChatConfiguration.queueStyle is set to
        // EstimatedWaitTime. Estimate is returned
        // in seconds.
    }
    else {
        // TO DO: Disable button or warn user that no agents are available
    }
}
})

```

In Objective-C:

```

SCSChatConfiguration *config =
    [[SCSChatConfiguration alloc] initWithLiveAgentPod:@"YOUR-POD-NAME"
                                                    orgId:@"YOUR-ORG-ID"
                                                    deploymentId:@"YOUR-DEPLOYMENT-ID"
                                                    buttonId:@"YOUR-BUTTON-ID"];

[[SCServiceCloud sharedInstance].chatCore
    determineAvailabilityWithConfiguration:config
    completion:^(NSError *error, BOOL available,
                  NSTimeInterval estimatedWaitTime) {

    if (error != nil) {
        // TO DO: Handle error
    }
    else if (available) {
        // TO DO: Enable chat button...

        // Optionally, use the estimatedWaitTime to
        // show an estimated wait time until an agent
        // is available. This value is only valid if
        // SCSChatConfiguration.queueStyle is set to
        // EstimatedWaitTime. Estimate is returned
        // in seconds.
    }
    else {
        // TO DO: Disable button or warn user that 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](#).

Transfer File to Agent

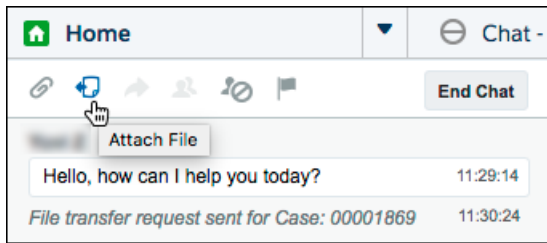
Give users the ability to transfer files during a chat so they can share information about their issues.



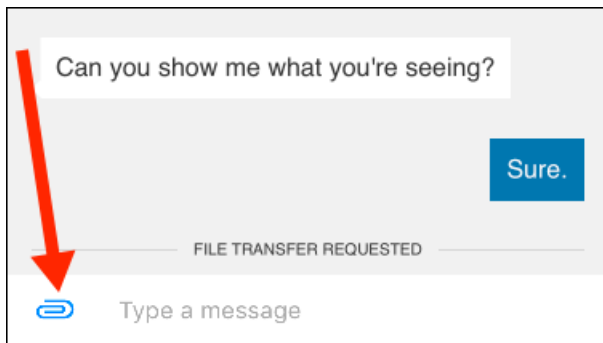
Important: The legacy chat product is scheduled for retirement on February 14, 2026, and is in maintenance mode until then. During this phase, you can continue to use chat, but we no longer recommend that you implement new chat channels. To avoid service interruptions to your customers, migrate to [Messaging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




The user sees a **FILE TRANSFER REQUESTED** message in the app and can then send a file using the paperclip button.




No coding is necessary in your app to make this behavior work.

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

 **Note:** If your app crashes when a user attempts to perform a file transfer, check that you've enabled the device privacy permissions for the camera and the photo library. An app will crash if these permissions are not set in Xcode. See [Install the Service Chat SDK for iOS](#).

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 running through these steps, be sure you've already:

- Set up Service Cloud to work with Chat. To learn more, see [Org Setup for Chat in Lightning Experience with a Guided Flow](#).
- Installed the SDK. To learn more, see [Install the Service Chat SDK for iOS](#).

Once you've reviewed these prerequisites, you're ready to begin.

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](#).

These steps describe how to use the Chat Core API with your own custom UI. To use the default UI, see [Quick Setup: Chat in the Service Chat SDK](#).

1. Import the SDK. Wherever you intend to use the Chat SDK, be sure to import the ServiceCore framework and the ServiceChat framework.

In Swift:

```
import ServiceCore
import ServiceChat
```

In Objective-C:

```
@import ServiceCore;
#import ServiceChat;
```

2. Create an [SCSChatConfiguration](#) object.

In Swift:

```
let config = SCSChatConfiguration(liveAgentPod: "TO_DO_POD_NAME",
                                // e.g. "d.gla5.gus.salesforce.com"
                                orgId: "TO_DO_ORG_ID",
                                // e.g. "00DB00000003Rxx"
                                deploymentId: "TO_DO_DEPLOYMENT_ID",
                                // e.g. "573B00000005KXz"
                                buttonId: "TO_DO_BUTTON_ID")
                                // e.g. "575C00000004h3m"
```

In Objective-C:

```
SCSChatConfiguration *config =
    [[SCSChatConfiguration alloc] initWithLiveAgentPod:@"TO_DO_POD_NAME"
                                                // e.g. "d.gla5.gus.salesforce.com"
                                                orgId:@"TO_DO_ORG_ID"
                                                // e.g. "00DB00000003Rxx"
                                                deploymentId:@"TO_DO_DEPLOYMENT_ID"
                                                // e.g. "573B00000005KXz"
                                                buttonId:@"TO_DO_BUTTON_ID"];
                                                // e.g. "575C00000004h3m"
```

See [Configure a Chat Session](#) on how to configure a chat session.

3. Implement `SCSChatSessionDelegate` and handle the relevant session-related methods.

Using this delegate, you can:

- Detect state transitions with `session (didTransitionFrom:to:)`.
- Detect the end of the session with `session (didEnd:)`.
- Detect error conditions with `session (didError:fatal:)`.
- Detect queue state changes with `session (didUpdateQueuePosition:estimatedWaitTime:)`.

Pass your implementation to the `SCSChat` instance.

In Swift:

```
ServiceCloud.shared().chatCore.add(delegate: mySessionDelegate)
```

In Objective-C:

```
[[SCServiceCloud sharedInstance].chatCore addDelegate:mySessionDelegate];
```

To learn more, see [Listen for State Changes and Events](#).

4. Implement `SCSChatEventDelegate` and handle the relevant event-related methods.

Using this delegate, you can:

- Detect when an agent joins with `session (agentJoined:)`.
- Detect when an agent leaves with `session (agentLeftConference:)`.
- Detect when an outgoing message is sent with `session (processedOutgoingMessage:)`.
- Detect when the delivery status of a message has been updated with `session (didUpdateOutgoingMessageDeliveryStatus:)`.
- Detect when an incoming message arrives with `session (didReceiveMessage:)`.
- Detects when a URL is found in an message with `session (didReceiveURL:)`.
- Detect when a chat bot menu arrives with `session (didReceiveChatBotMenu:)`.
- Detect when a chat bot menu item is selected with `session (didSelectMenuItem:)`.
- Detect when the agent starts and finishes typing.
- Detect events related to the file transfer process.
- Detects when the user is transferred to an agent from a chat bot.

Pass your implementation to the `SCSChat` instance.

In Swift:

```
ServiceCloud.shared().chatCore.addEvent(delegate: myEventDelegate)
```

In Objective-C:

```
[[SCServiceCloud sharedInstance].chatCore addEventDelegate:myEventDelegate];
```

5. Start the session using `startSession (with:)` on `SCSChat`.

In Swift:

```
ServiceCloud.shared().chatCore.startSession(config)
```

In Objective-C:

```
[[SCServiceCloud sharedInstance].chatCore startSessionWithConfiguration:config];
```

6. Send activity to the `SCSChatSession` object found in `SCSChat`.


You can access the session object either from the `SCSChat.session` property, or from any of your delegate event methods that the SDK calls.

With this session object, you can:

- Send sneak peek data about the user's message to the agent with `sendSneakPeek`.
- Send a message to the agent with `sendMessage`.
- Get or set the user's typing status with `userTyping`.
- Get information about the actors in the chat session with `actors`.
- Get the history of all events from the chat session with `allEvents`.
- Get the current queue position when the user is waiting for an agent with `queuePosition`.

SDK Customizations with the Service Chat SDK for iOS

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

Many UI customizations are handled with the `SCAppearanceConfiguration` object. You can configure the colors, fonts, and images to your interface with an `SCAppearanceConfiguration` instance. It contains the methods `setColor`, `setFontDescriptor`, and `setImage`. To use this object, create an `SCAppearanceConfiguration` instance, specify values for each token you want to change, and store the instance in the `appearanceConfiguration` property of the `ServiceCloud sharedInstance`.

There are other ways to customize the interface. When using Service Cloud features, various action buttons are available to the user. You can control the visibility of these buttons and even create new action buttons. You can also customize the strings used in the UI for any of the supported languages. String customization is performed using a standard [localization mechanism](#) provided to Apple developers.

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

Customize the colors by defining the branding token colors used throughout the interface.

[Customize and Localize Strings with the Service Chat SDK](#)

You can change the text used throughout the user interface. To customize text, create string resource values in a `Localizable.strings` file in the Localization bundle for the languages you want to update. Create string tokens that match the tokens you intend to override.

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

There are three customizable font settings used throughout the UI: `SCFontWeightLight`, `SCFontWeightRegular`, `SCFontWeightBold`.

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

You can specify custom images used throughout the UI.

Customize Colors with the Service Chat SDK

Customize the colors by defining the branding token colors used throughout the interface.



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 colors, create an `SCAppearanceConfiguration` instance, specify values for each token you want to change, and store the instance in the `appearanceConfiguration` property of the `ServiceCloud.sharedInstance`.

In Swift:

```
// Create appearance configuration instance
let appearance = SCAppearanceConfiguration()

// Customize color tokens
appearance.setColor(COLOR_VALUE, forName: TOKEN_NAME)

// Add other customizations here...

// Save configuration instance
ServiceCloud.shared().appearanceConfiguration = appearance
```

In Objective-C:





```
// Create appearance configuration instance
SCAppearanceConfiguration *appearance = [SCAppearanceConfiguration new];



















// Customize color tokens
[appearance setColor:COLOR_VALUE forName:TOKEN_NAME];








// Add other customizations here...

// Save configuration instance
[SCServiceCloud sharedInstance].appearanceConfiguration = appearance;
```

To support dark mode in iOS 13 and later, [specify adaptive colors](#) for each branding token. The following branding tokens are available for customization.

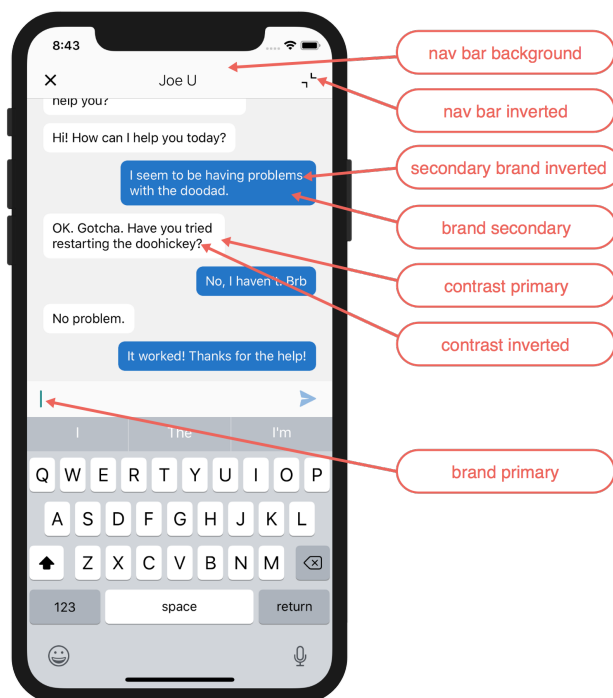
Token Name / Swift Value / Objective-C Value	Default	Dark Mode Default	Description / Sample Uses
Navigation Bar Background <code>navbarBackground</code> <code>SCAppearanceColorDefaultBackground</code>	#FAFAFA 	#1A2129 	Background color for the navigation bar.
Navigation Bar Inverted <code>navbarInverted</code> <code>SCAppearanceColorDefaultInverted</code>	#010101 	#C6CBCF 	Navigation bar text and icon color.

Token Name / Swift Value / Objective-C Value	Default	Dark Mode Default	Description / Sample Uses
Brand Primary <code>brandPrimary</code> <code>SSAppearanceColorOfTheBrandPrimary</code>	#007F7F 	#00B4B4 	
Brand Secondary <code>brandSecondary</code> <code>SSAppearanceColorOfTheBrandSecondary</code>	#2872CC 	#0070D2 	Used throughout the UI for button colors. Chat: Agent text bubbles.
Primary Brand Inverted <code>brandPrimaryInverted</code> <code>SSAppearanceColorOfTheBrandPrimaryInverted</code>	#FBFBFB 	#FBFBFB 	
Secondary Brand Inverted <code>brandSecondaryInverted</code> <code>SSAppearanceColorOfTheBrandSecondaryInverted</code>	#FCFCFC 	#F7F7F7 	Text on areas where a brand color is used for the background.
Contrast Primary <code>contrastPrimary</code> <code>SSAppearanceColorOfTheContrastPrimary</code>	#000000 	#E2E4E6 	Primary body text color.
Contrast Secondary <code>contrastSecondary</code> <code>SSAppearanceColorOfTheContrastSecondary</code>	#6D6D6D 	#898D92 	
Contrast Tertiary <code>contrastTertiary</code> <code>SSAppearanceColorOfTheContrastTertiary</code>	#BABABA 	#A0A6AD 	
Contrast Quaternary <code>contrastQuaternary</code> <code>SSAppearanceColorOfTheContrastQuaternary</code>	#F1F1F1 	#09121B 	Chat: Background color.
Contrast Inverted <code>contrastInverted</code> <code>SSAppearanceColorOfTheContrastInverted</code>	#FFFFFF 	#323232 	Page background, navigation bar, table cell background.

Token Name / Swift Value / Objective-C Value	Default	Dark Mode Default	Description / Sample Uses
Feedback Primary <code>feedbackPrimary</code> <code>SCAppearanceColorTokenFeedbackPrimary</code>	#E74C3C 	#E0A7A9 	Text color for error messages.
Feedback Secondary <code>feedbackSecondary</code> <code>SCAppearanceColorTokenFeedbackSecondary</code>	#2ECC71 	#9ACDB7 	
Feedback Tertiary <code>feedbackTertiary</code> <code>SCAppearanceColorTokenFeedbackTertiary</code>	#F5A623 	#FADBAE 	
Overlay <code>overlay</code> <code>SCAppearanceColorTokenOverlay</code>	Contrast Primary (at 40% alpha)	#323232 	

These screenshots illustrate how the branding tokens affect the UI.

Chat UI Branding:





Example: The following code sample changes three of the branding tokens.

In Swift:

```
// Create appearance configuration instance
let appearance = SCAppearanceConfiguration()

// Customize color tokens
appearance.setColor(
    UIColor(red: 80/255, green: 227/255, blue: 194/255, alpha: 1.0),
    forName: .brandPrimary)
appearance.setColor(
    UIColor(red: 74/255, green: 144/255, blue: 226/255, alpha: 1.0),
    forName: .brandSecondary)
appearance.setColor(
    UIColor(red: 252/255, green: 252/255, blue: 252/255, alpha: 1.0),
    forName: .brandSecondaryInverted)

// Save configuration instance
ServiceCloud.shared().appearanceConfiguration = appearance
```

In Objective-C:

```
// Create appearance configuration instance
SCAppearanceConfiguration *appearance = [SCAppearanceConfiguration new];


// Customize color tokens
[appearance setColor:[UIColor colorWithRed: 80/255
                                     green: 227/255
                                     blue: 194/255
                                     alpha: 1.0]
                 forName:SCSAppearanceColorTokenBrandPrimary];
[appearance setColor:[UIColor colorWithRed: 74/255
                                     green: 144/255
                                     blue: 226/255
                                     alpha: 1.0]
                 forName:SCSAppearanceColorTokenBrandSecondary];
[appearance setColor:[UIColor colorWithRed: 252/255
                                     green: 252/255
                                     blue: 252/255
                                     alpha: 1.0]
                 forName:SCSAppearanceColorTokenBrandSecondaryInverted];

// Save configuration instance
[SCServiceCloud sharedInstance].appearanceConfiguration = appearance;
```

For an example of how to specify adaptive colors that work with the dark mode feature that was introduced in iOS 13, see [Handling Dark Mode for iOS 13 with the Service SDK](#).

Customize and Localize Strings with the Service Chat SDK

You can change the text used throughout the user interface. To customize text, create string resource values in a `Localizable.strings` file in the Localization bundle for the languages you want to update. Create string tokens that match the tokens you intend to override.

 **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](#).

Service Chat SDK text is translated into more than 25 different languages. In order for your string customizations to take effect in a given language, provide a translation for that language. For any language you do not override manually in your app, the SDK uses its default values for that language.

Refer to [Internationalization at developer.apple.com](https://developer.apple.com/internationalization) for more info about localization.

The following list of string tokens are available for customization:

- [ServiceChat \(Chat\) String Resources](#)
- [ServiceCore \(Common\) String Resources](#)

The following languages are currently supported:


Table 5: Supported Languages

Language Code	Language
ar	Arabic
cs	Czech
da	Danish
de	German
el	Greek
en	English
es	Spanish
fi	Finnish
fr	French
hu	Hungarian
id	Indonesian
it	Italian
iw	Hebrew
ja	Japanese
ko	Korean
nb	Norwegian Bokmål
nl	Dutch
pl	Polish
pt	Portuguese

Language Code	Language
ro	Romanian
ru	Russian
sv	Swedish
th	Thai
tr	Turkish
uk	Ukrainian
vi	Vietnamese
zh_TW	Chinese (Taiwan)
zh-Hans	Chinese (Simplified)
zh-Hant	Chinese (Traditional)
zh	Chinese

Customize Fonts with the Service Chat SDK

There are three customizable font settings used throughout the UI: `SCFontWeightLight`, `SCFontWeightRegular`, `SCFontWeightBold`.

 **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](#).

You can customize three font settings used throughout the Service Chat SDK interface:

Font Setting	Default Value	Samples Uses in the SDK
<code>SCFontWeightLight</code>	Helvetica Neue - Light	Knowledge article cell summary, Case Management field text, Case Management submit success view, content of error messages
<code>SCFontWeightRegular</code>	Helvetica Neue	Navigation bar, Chat text, Knowledge data category cell in detail view, Knowledge "show more" article footer, Knowledge "show more" button cell
<code>SCFontWeightBold</code>	Helvetica Neue - Semibold	Knowledge category headers, Knowledge article cell title, Case Management field labels, Case Management submit button, title of error messages

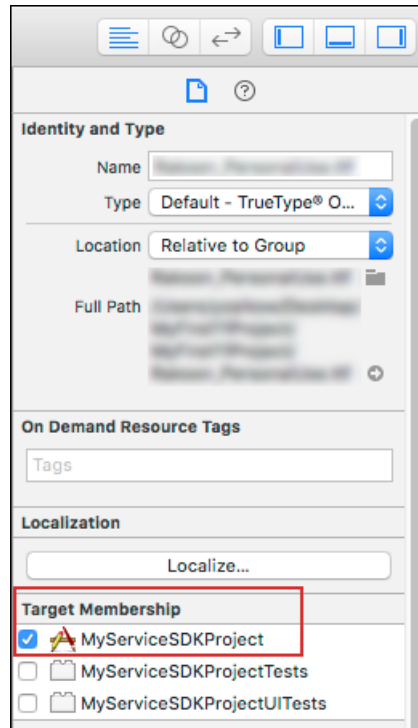
To configure your app to use different fonts:

1. Add new fonts to your Xcode project.

Any new fonts must be added as a resource to your Xcode project. When adding, be sure to select **Copy items if needed**.

2. Add new fonts to your project target.

For each new font, add it to your project target under **Target Membership**.



3. Add the font to your app's `Info.plist`.

You'll need to add all new fonts into a string array. Each string element of the array must be the name of each font resource file.

If you're viewing your `Info.plist` as a **Property List**, add an Array named **Fonts provided by application**.

If you're viewing your `Info.plist` as **Source Code**, add an array named `UIAppFonts`. For example:

```
<key>UIAppFonts</key>
<array>
  <string>MyCustomFont1.ttf</string>
  <string>MyCustomFont2.ttf</string>
  <string>MyCustomFont3.ttf</string>
</array>
```

4. Customize any of the Service Chat SDK font values using `SCAppearanceConfiguration`.

To customize the fonts, create an `SCAppearanceConfiguration` instance, set the font descriptor for each font setting you want to change, and store the `SCAppearanceConfiguration` instance in the `appearanceConfiguration` property of the `ServiceCloud` shared instance.

Swift Example:

```
// Create appearance configuration instance
let config = SCAppearanceConfiguration()

// Customize font
let descriptor = UIFontDescriptor(fontAttributes:
    [UIFontDescriptor.AttributeName.family : "Proxima Nova"])
config.setFontDescriptor(descriptor,
    fontFileName: "ProximaNova-Light.otf",
    forWeight: SCFontWeightLight)

// Add other customizations here...

// Save configuration instance
ServiceCloud.shared().appearanceConfiguration = config
```

Objective-C Example:

```
// Create appearance configuration instance
SCAppearanceConfiguration *config =
    [SCAppearanceConfiguration new];

// Customize font
UIFontDescriptor *descriptor =
    [UIFontDescriptor fontDescriptorWithFontAttributes:@{
        UIFontDescriptorFamilyAttribute: @"Proxima Nova",
        UIFontDescriptorFaceAttribute: @"Light" }];
[config setFontDescriptor:descriptor
    fontFileName:@"ProximaNova-Light.otf"
    forWeight:SCFontWeightLight];


// Add other customizations here...

// Save configuration instance
[SCServiceCloud sharedInstance].appearanceConfiguration = config;
```

Be sure to use the exact font descriptor attribute name and font file name for your custom font.

Customize Images with the Service Chat SDK

You can specify custom images 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](#).

Use the [setImage](#) method on the [SCAppearanceConfiguration](#) object to **replace a stock image** with your image. Use the enumeration value for the image you intend to replace.

Supported image file formats include: tiff, tif, jpg, jpeg, gif, png, bmp, BMPF, ico, cur.

For specific images, use the [SCSAppearanceImageToken](#) enumeration specified by the SDK and add it to the [SCAppearanceConfiguration](#) object with the [setImage](#) method.

Table 6: Stock Image Enum Values

Image Description	Enum Value
Close button	close
Done button	done
Small warning icon used when an error occurs	error
Error image used in a view for timeouts, when no agents are available, or for an unknown error	genericError
Minimize button (Knowledge and Chat)	minimizeButton
No connection	noConnection
Send button (Case Publisher and Chat)	send
Next field button (Case Publisher and Chat)	submitButtonNextArrow
Previous field button (Case Publisher and Chat)	submitButtonPreviousArrow
Attachment button when the user can attach a file	attachmentClipIcon
Avatar used for the agent (only the inscribed circle is visible in the chat feed)	chatAgentAvatar
Avatar used for Einstein bot (only the inscribed circle is visible in the chat feed)	chatBotAvatar
Icon used for Einstein bot persistent footer menu	chatBotFooterMenu
Icon used in the pre-chat screen	preChatIcon

In Swift:

```
// Create appearance configuration instance
let config = SCAppearanceConfiguration()

// Specify images
config.setImage(MY_CUSTOM_IMAGE,
               compatibleWithTraitCollection: MY_TRAITS,
               forName: ENUM_VALUE)

// Add other customizations here...

// Save configuration instance
ServiceCloud.shared().appearanceConfiguration = config
```

In Objective-C:

```
// Create appearance configuration instance
SCAppearanceConfiguration *config = [SCAppearanceConfiguration new];

// Specify images
[config setImage:MY_CUSTOM_IMAGE compatibleWithTraitCollection: MY_TRAITS
```

```
        forName: ENUM_VALUE];

// Add other customizations here...

// Save configuration instance
[SCServiceCloud sharedInstance].appearanceConfiguration = config;
```

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 iOS SDK](#)

To configure the Service Chat SDK logs, set the `level` property on the `ServiceLogger` shared instance.

[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.

[Agent Availability Check Always Fails](#)

What to look into when agent availability fails.

[My App Crashes](#)

Some tips if your app crashes.

[My App Was Rejected](#)

What to do when your app is rejected from the App Store.

Enable Debug Logging for the iOS SDK

To configure the Service Chat SDK logs, set the `level` property on the `ServiceLogger` shared instance.

Use the `shared` singleton on `ServiceLogger` to adjust the `level`.

In Swift:

```
ServiceLogger.shared.level = .debug
```

In Objective-C:

```
[SCServiceLogger sharedInstance].level = SCLoggerLevelDebug;
```

The log level is specified using the `SCLoggerLevel` enumerated type. It can be one of these values:

- `SCLoggerLevelDebug` (`.debug` in Swift)
- `SCLoggerLevelInfo` (`.info` in Swift)
- `SCLoggerLevelError` (`.error` in Swift)—Default value
- `SCLoggerLevelFault` (`.fault` in Swift)

- `SCSLoggerLevelOff` (`.off` in Swift)

By default, logs go to the console output. You can have logs go to a file using the `filehandle` property.

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

Agent Availability Check Always Fails

What to look into when agent availability fails.

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.

My App Crashes

Some tips if your app crashes.

- Chat: If your app crashes when a user attempts to perform a file transfer, check that you've enabled the device privacy permissions for the camera and the photo library. An app will crash if these permissions are not set in Xcode. See [Install the Service Chat SDK for iOS](#).
- For a list of known issues, see the latest [Release Notes](#).


My App Was Rejected

What to do when your app is rejected from the App Store.

- If you archive a framework and then export the archive using the Xcode command line tool (`xcodebuild`), you'll get "Invalid Code Signing Entitlements" errors when you try to upload your app to the app store. This is a known issue with Apple's tools. The workaround is to archive and export using Xcode's user interface.


Data Protection and Security in the Service Chat SDK for iOS

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 iOS Keychain Services. All content fetched from Salesforce servers is stored locally using AES-128 encryption. When the user logs out, we remove all user-specific data from the device.
 - **Secure data in transit.** All network communication occurs over SSL using TLS 1.2.

Reference Documentation

Reference documentation for Service Chat SDK for iOS.

-  **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 iOS, visit:

- forcedotcom.github.io/ServiceSDK-iOS


This site contains API documentation for the latest version of the SDK.

[Reference Index](#)

A list of all classes, protocols, methods, constants, and enums referenced from this developer's guide.

Reference Index

A list of all classes, protocols, 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

- [SCSChat](#)
 - [add\(delegate:\)](#)
 - [addEvent\(delegate:\)](#)
 - [determineAvailabilityWithConfiguration](#)
 - [startSession\(with:\)](#)
 - [startSession\(with:completion:\)](#)

- `SCSChatInterface`
 - `handle(notification:)`
 - `showChat(with:showPrechat:)`
 - `showPrechat(withFields:modal:completion:)`
 - `shouldDisplayNotificationInForeground`
- `SCSChatInterfaceDelegate`
- `SCSChatConfiguration`
 - `maximumEstimatedWaitTime`
 - `minimumEstimatedWaitTime`
 - `prechatEntities`
 - `prechatFields`
 - `queueStyle`
- `SCSChatEventDelegate`
 - `session(agentJoined:)`
 - `session(agentLeftConference:)`
 - `session(didReceiveChatBotMenu:)`
 - `session(didReceiveFileTransferRequest:)`
 - `session(didReceiveMessage:)`
 - `session(didReceiveURL:)`
 - `session(didSelectMenuItem:)`
 - `session(didUpdateOutgoingMessageDeliveryStatus:)`
 - `session(processedOutgoingMessage:)`
- `SCSChatSessionDelegate`
 - `session(didEnd:)`
 - `session(didError:fatal:)`
 - `session(didUpdateQueuePosition:estimatedWaitTime:)`
 - `session(didTransitionFrom:to:)`
- `SCSChatEndReason`
- `SCSChatErrorCode`
- `SCSChatSession`
- `SCSChatSessionState`
- `SCSPrechatEntity`
- `SCSPrechatEntityField`
- `SCSPrechatObject`
- `SCSPrechatPickerObject`
- `SCSPrechatPickerOption`
- `SCSPrechatTextInputObject`

Service Common Index

- `SCAppearanceConfiguration`
 - `globalArticleCSS`
 - `globalArticleJavascript`
 - `setColor`
 - `setFontDescriptor`
 - `setImage`
- `SCAppearanceConfigurationDelegate`
- `SCSAppEventList`
- `SCSActionButton`
- `SCSActionItem`
- `SCSActionItemContainer`
- `SCSActionManager`
- `SCSActionManagerDelegate`
- `SCSAuthenticationSettings`
- `ServiceCloud`
 - `actions`
 - `appearanceConfiguration`
 - `cases`
 - `chatCore`
 - `chatUI`
 - `knowledge`
 - `notification(fromRemoteNotificationDictionary:)`
 - `setAuthenticationSettings(settings:forServiceType:completion:)`
 - `sharedInstance`
 - `showInterface(for:)`
 - `sos`
- `SCServiceCloudDelegate`
 - `serviceCloud(didDisplay controller:animated:)`
 - `serviceCloud(authenticationFailed:forServiceType:)`
 - `serviceCloud(shouldAuthenticateServiceType:completion:)`
 - `serviceCloud(transitioningDelegateForPresentedController:presenting:)`
 - `serviceCloud(willDisplay controller:animated:)`
- `SCSServiceConfiguration`
 - `imageFolderPath`
 - `SCSServiceConfiguration(community:)`
 - `SCSServiceConfiguration(community:dataCategoryGroup:rootDataCategory:)`
- `SCSNotification`

- `SCSNotificationType`
- `ServiceLogger`
 - `filehandle`
 - `level`
 - `shared`

Resource Files

- `ServiceChat (Chat) String Resources`
- `ServiceCore (Common) String Resources`

Deprecated APIs

- `SCArticleSortByField`
- `SCArticleSortOrder`
- `SCKnowledgeInterface`
 - `setInterfaceVisible`
 - `showArticle`
- `SCKnowledgeInterfaceDelegate`
- `SCQueryMethod`
- `Article`
 - `downloadContent (withOptions:)`
 - `isArticleContentDownloaded`
 - `isAssociatedContentDownloaded`
- `SCSArticleQuery`
 - `valid`
- `SCSArticleQueryListViewController`
- `SCSArticleQueryListViewControllerDelegate`
- `SCSArticleViewController`
 - `article`
- `SCSArticleViewControllerDelegate`
 - `additionalCSSForArticle`
 - `additionalJavascriptForArticle`
- `Category`
- `CategoryGroup`
- `SCSCategoryViewController`
- `SCSCategoryViewControllerDelegate`
- `SCSKnowledgeHomeViewController`
- `SCSKnowledgeHomeViewControllerDelegate`

- `KnowledgeManager`
 - `articles(matching:)`
 - `defaultManager`
 - `fetchAllCategories`
 - `fetchArticles(with:)`
 - `hasFetchedCategories`
- `MutableArticleQuery`
- `SCSCaseInterface`
 - `caseCreateActionName`
 - `setInterfaceVisible`
- `SCSCaseDetailViewController`
- `SCSCaseDetailViewControllerDelegate`
 - `caseDetail(fieldsToHideFromCaseFields:)`
- `SCSCaseListViewController`
- `SCSCaseListViewControllerDelegate`
 - `caseList(selectedCaseWithId:)`
- `SCSCasePublisherViewController`
- `SCSCasePublisherViewControllerDelegate`
 - `casePublisher(fieldsForCaseDeflection:)`
 - `casePublisher(fieldsToHideFromCaseFields:)`
 - `casePublisher(valuesForHiddenFields:)`
 - `casePublisher(viewFor:withCaseId:error:)`
 - `shouldEnableCaseDeflection(forPublisher:)`
- `SOSAgentAvailability`
 - `startPolling(withOrganizationId:deploymentId:liveAgentPod:)`
- `SOSAgentAvailabilityDelegate`
- `SOSAgentAvailabilityStatusType`
- `SOSCameraType`
- `SOSConnectingBaseViewController`
- `SOSConnectingViewController`
- `SOSDelegate`
 - `sosDidStart`
 - `sosDidConnect`
 - `sosWillReconnect`
 - `sos(didCreateSession:)`
 - `sos(didError:)`
 - `sos(didStopWith:error:)`

- `sos (stateDidChange:current:previous:)`
- `SOSErrorCode`
- `SOSMaskedTextField`
- `SOSNetworkReporterDelegate`
- `SOSOnboardingBaseViewController`
- `SOSOnboardingViewController`
- `SOSOptions`
 - `customFieldData`
 - `featureAgentVideoStreamEnabled`
 - `featureClientBackCameraEnabled`
 - `featureClientFrontCameraEnabled`
 - `featureClientScreenSharingEnabled`
 - `featureNetworkTestEnabled`
 - `initialAgentStreamPosition`
 - `initialAgentVideoStreamActive`
 - `initialCameraType`
 - `remoteLoggingEnabled`
 - `sessionRetryTime`
 - `setViewControllerClass`
 - `SOSOptions (liveAgentPod:orgId:deploymentId:)`
- `SOSSessionBaseViewController`
- `SOSSessionViewController`
- `SOSScreenSharingBaseViewController`
- `SOSSessionManager`
 - `add(delegate: SOSDelegate!)`
 - `screenSharing`
 - `startSession`
 - `state`
 - `stopSession`
 - `stopSession(completion:)`
- `SOSSessionState`
- `SOSStopReason`
- `SOSUIAgentStreamReceivable`
- `SOSUILineDrawingReceivable`
- `SOSUIPhase`
- `ServiceKnowledge (Knowledge) String Resources`
- `ServiceCases (Case Management) String Resources`
- `ServiceSOS (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

app store submission [23](#)
authentication [18](#)

B

branding [64](#)

C

color customization [65](#)

D

data protection [75](#)
dynamic libraries [23](#)

E

Einstein bots [37](#)

F

font customization [70](#)
fonts [68](#)

I

image customization [72](#)
install sdk [15–17](#)

L

logging in iOS [74](#)

P

prerequisites [13](#)

push notifications [22](#)

Q

quick start [24](#)

R

reference index [76](#)
reference overview [76](#)
release notes [2](#)
remote notifications [22](#)

S

sdk install [15–17](#)
SDK prerequisites [13](#)
sdk setup [12](#)
service cloud setup [2](#)
setup [2, 12](#)
stock images [72](#)

T

troubleshooting
 app store [75](#)
 session start [75](#)

U

ui customization
 colors [65](#)
 fonts [68, 70](#)
 images [72](#)