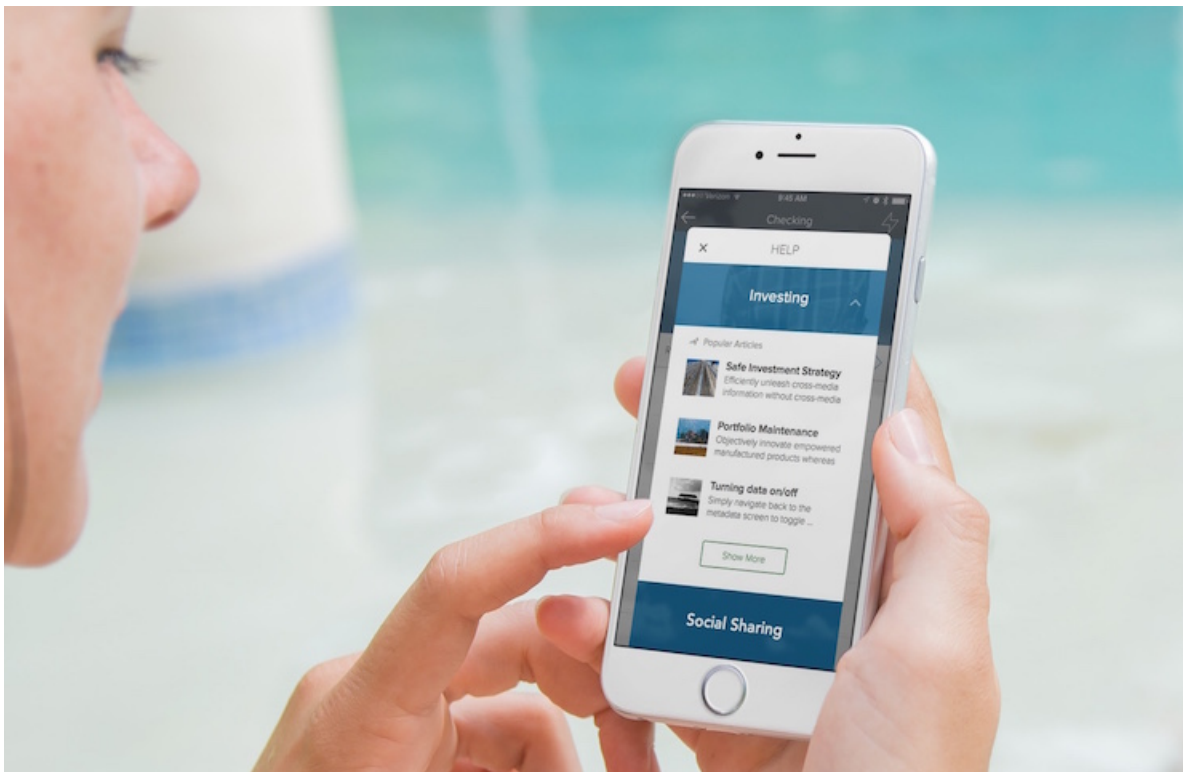




Service SDK Developer's Guide for iOS

Version 2.0.2



CONTENTS

Service SDK Developer's Guide for iOS	1
Release Notes	2
Service Cloud Setup	6
SDK Setup	27
Quick Start Tutorials	33
Using Knowledge	51
Using Case Management	71
Using Live Agent Chat	93
Using SOS	103
SDK Customizations	131
Troubleshooting	153
Reference Documentation	160
Additional Resources	298
Index	299

SERVICE SDK DEVELOPER'S GUIDE FOR IOS

The Service SDK for mobile devices makes it easy to give customers access to powerful Service Cloud features right from within your native mobile app. You can make these Service Cloud elements feel organic to your app and have things up and running quickly. This developer's guide helps you get started with the Service SDK.

SDK Version 2.0.2.

This documentation describes the following component versions, which are packaged together in this version of the SDK:

Component	Version Number
Knowledge	2.0.0
Case Management	1.2.0
Live Agent Chat	1.1.1
SOS	3.2.2
ServiceCore (common component used by all features)	1.2.0

[Release Notes](#)

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

[Service Cloud Setup](#)

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

[SDK Setup](#)

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

[Quick Start Tutorials](#)

Get going quickly with these short introductory tutorials.

[Using Knowledge](#)

Adding the Knowledge experience to your app.

[Using Case Management](#)

Adding the Case Management experience to your app.

[Using Live Agent Chat](#)

Adding the Live Agent Chat experience to your app.

[Using SOS](#)

Adding the SOS experience to your app.

[SDK Customizations](#)

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

[Troubleshooting](#)

Get some guidance when you run into issues.

[Reference Documentation](#)

Reference documentation for the Service SDK.

[Additional Resources](#)

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

Release Notes

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

Version 2.0.2 (12/15/2016)

Fixes:

- SOS: Fixed an issue where leaving a secure text field doesn't resume screen sharing.

Version 2.0.1 (12/2/2016)

New Features:

- SOS: Updated to version 2.9.2 of OpenTok.
- SOS: Added logging for Quality of Service (QOS) information from OpenTok on audio and video streams to Splunk.

Fixes:

- SOS: Fixed an issue where the 'Screen Sharing Disabled' notification was not disappearing when screen sharing is turned back on and the user mic is muted.
- SOS: Upgraded all OpenTok errors to be uniform with Widget and Android.
- Live Agent Chat: Fixed the issue where the UI wouldn't appear when launching a session in non-blocking mode without any pre-chat fields.
- Live Agent Chat: Fixed the issue where availability checks caused the SDK to crash.

Version 2.0.0 (11/10/2016)

New Features:

- Knowledge: Users can view specific knowledge bases associated with their user profile using authenticated users. To learn more, see [Knowledge as an Authenticated User](#).
- Knowledge: You can display specific articles using `SCSArticleViewController`. To learn more, see [Customize the Presentation for Knowledge](#).
- Case Management: You can deflect case creation by pointing users to related knowledge base articles. To learn more, see [Configure Case Deflection](#).
- Case Management: Message bubbles support tappable hyperlinks. Text can be copied with a long press.
- Live Agent Chat: You can check for agent availability before attempting to start a session. To learn more, see [Check Live Agent Availability](#).
- Live Agent Chat: You can use a dropdown list as a pre-chat field using `SCSPrechatPickerObject`. To learn more, see [Configure a Live Agent Chat Session](#).
- Live Agent Chat: Message bubbles support tappable hyperlinks. Text can be copied with a long press.

- SOS: Image branding now supported. To learn more, see [Customize Images](#).
- SOS: You can programmatically enable and disable screen sharing. To learn more, see [Enable and Disable Screen Sharing](#).
- SOS: Updated to [OpenTok](#) version 2.9.1.
- SOS: SOS now supports elegant reconnects when there are network issues.
- SOS: UI displays when the session is disabled.
- SOS: Added a value to the [SOSOptions](#) object that allows users to specify a starting location for the agent stream (`initialAgentStreamPosition`).
- SOS: Added the VoiceOver accessibility feature to SOS.

API Changes:

- Knowledge: [SCSArticleViewController](#) added to display specific articles.
- Knowledge: [SCSMutableArticleQuery](#) added for use when creating a query.
- Knowledge: [SCSArticleQuery](#) is immutable.
- Case Management: [SCSCasePublisherViewControllerDelegate](#) now supports methods for case deflection.
- Live Agent Chat: [SCSPrechatPickerObject](#) and [SCSPrechatPickerOption](#) added for use with a pre-chat field.
- SOS: [SOSScreenSharing](#) added for enabling and disabling screen sharing. A `screenSharing` property has also been added to [SOSSessionManager](#).
- SOS: [SOSOptions](#) added `initialAgentStreamPosition` property to set the default position of the agent stream.
- SOS: [SOSUIAgentStreamReceivable](#) can tell you when the recording view has changed.

Fixes:

- Knowledge: Fixed the issue where minimized article view does not appear after Case Publisher is closed.
- Case Management: Fixed an issue where tapping on a case list item could crash the app.
- Case Management: Fixed an issue where the case detail view crashed when loading content.
- Live Agent Chat: Fixed the issue where the user end session alert would not be dismissed when an agent ends a session while the alert is visible.
- Live Agent Chat: Fixed the issue where Chat occasionally forced status bar style change.
- Live Agent Chat: Fixed the issue where starting another session mid-session was destructive to ongoing session.
- SOS: Fixed an issue where agent availability was not setting the state to unknown when a error occurred.
- SOS: Fixed an issue where the mute icon is not shown in the camera view after the flashlight icon is shown.
- SOS: Fixed an issue where the SDK ends a session when the widget temporarily disconnects.
- SOS: Fixed an issue where the `agentName` background view is wrong when loading the UI.
- SOS: Fixed an issue where the `recordingView` was not displaying properly or working with branding.

Known Issues:

- When you customize the appearance using a new instance of `SCAppearanceConfiguration` and assign it to the `SCServiceCloud.appearanceConfiguration` property, the view controller becomes invisible if it's pushed (rather than presented modally). **Workaround:** You can avoid this problem by using the existing instance of `SCServiceCloud.appearanceConfiguration` instead of instantiating a new one.
- Although some accessibility features have been added to this version, the SDK does not yet support VoiceOver for Knowledge, Case Management, or Live Agent Chat.
- 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. **Workaround:** Archive and export using Xcode's user interface.

- An issue with Xcode 8 when using the iOS 10 simulator results in a build error about "adding a keychain item". This error has been filed with Apple as [radar #27422249](#). **Workaround:** Turn on **Keychain Sharing** in your build target's **Capabilities**.
- Knowledge: Images uploaded with [CKEditor](#) don't work for authenticated users when specifying a non-community URL for your Knowledge configuration. In that situation, images will not appear. **Workaround:** Use web links to inline images hosted elsewhere.
- Knowledge: When the same article is in multiple categories, the **Show More** button may not work properly. **Workaround:** Don't assign multiple categories to one article.
- Case Management: Case Publisher view fails to open in app when you add a **Blank Space** to the Case Action layout. **Workaround:** Remove **Blank Space** from the Case Action layout.
- Case Management: If you display the case detail view controller ([SCSCaseDetailViewController](#)) without assigning a case ID, the view will be a blank, black window. **Workaround:** Be sure to initialize the case detail view controller with the case ID using [initWithCaseId:](#).
- Case Management: Read-only fields in Case Action layouts display as editable fields when creating a new case using the SDK's Case Publisher. **Workaround:** Ensure that there are no read-only fields in the Case Action layout you use to create cases with the SDK.
- Live Agent Chat: The UI for a chat session doesn't appear when launching chat in non-blocking mode without any pre-chat fields. **Workaround:** Add pre-chat fields, or start session in full-screen mode. See [Configure a Live Agent Chat Session](#) for more information.
- SOS: The size of the agent placeholder icon (named "avatar") must be exactly 61x55 pixels (@1x), 122x110 (@2x), 183x165 (@3x). If you attempt to use a different resolution, the agent stream view will not be centered.

Version 1.2.0 (9/15/2016)

New Features:

- Case Management: Added the ability to customize the Case Publisher submission success view. To learn more, see [Customize the Case Publisher Result View](#).

API Changes:

- Case Management: [SCSCasePublisherViewControllerDelegate](#) updated with a new delegate method to support the new Case Publisher submission success view. To learn more, see [Customize the Case Publisher Result View](#).

Fixes:

- Knowledge: Fixed an issue where duplicate articles show up in the article list.
- Knowledge: Fixed an issue where searching for an article could cause the app to freeze.
- Knowledge: Fixed an issue where dismissing the interface fails to make the host application window the key window.
- Fixed a sporadic crash when assigning a user account with `SCServiceCloud.account`.
- Knowledge: Fixed an issue where article detail returns 'Unable to connect' when `SCServiceCloud.account` changes.
- SOS: Changed the SOS agent availability poll time from 5 seconds to 30 seconds.

Known Issues:

- Case Management: Read-only fields in Case Action layouts display as editable fields when creating a new case using the SDK's Case Publisher. **Workaround:** Ensure that there are no read-only fields in the Global Action layout you use to create cases with the SDK.
- Some accessibility issues still need to be resolved in Knowledge: showing interface does not announce title; unable to swipe to Contact and Search icon; unable to swipe to move out of Contact and Search icons; swiping from category focus does not work correctly.
- Some accessibility issues still need to be resolved in Case Management: unable to navigate into and out of navigation bar with swipe; can't move to next field with swipe in case publisher view; unable to use swipe to get to Create or Close button in case list view; unable to navigate into and out of text fields with swipe in case detail view; unable to navigate to Send button with swipe in case detail view.

- Knowledge: When fetching articles, `ViewScore` sorting does not work in this version. **Workaround:** Sort by `Title` or `LastPublishedDate` instead.
- Knowledge: When searching for articles with the search bar, articles in the root category are shown in the search results even though these articles don't show from the Knowledge interface (which only shows subcategories of the root category). **Workaround:** Avoid putting essential articles in the root category; use a subcategory instead.
- Knowledge: There is no current support for clearing the entire cache for articles. **Workaround:** Delete the app and reinstall.

Version 1.1.1 (9/2/2016)

- Minor update that improves compatibility with the Salesforce Mobile SDK when using a guest user for Case Management.

Version 1.1.0 (8/19/2016)

New Features:

- Knowledge, Case Management, and Live Agent Chat are Generally Available (formerly in Beta).
- Live Agent Chat supports file transfers.
- Each feature area is in a separate framework. See [Install the SDK](#) for installation instructions. And check out the **Quick Setup** section for each feature area for instructions on importing each feature: [Quick Setup: Knowledge](#), [Quick Setup: Case Publisher as a Guest User](#), [Quick Setup: Live Agent Chat](#), [Quick Setup: SOS](#).
- You can now hide fields in the Case Management views in order to send custom data. To learn more, see [Send Custom Data Using Hidden Fields](#).
- API changes:
 - [SCSCaseDetailViewControllerDelegate](#) updated to support hidden fields in Case Management. To learn more, see [Send Custom Data Using Hidden Fields](#).
 - [SCSCasePublisherViewControllerDelegate](#) updated to support hidden fields in Case Management. To learn more, see [Send Custom Data Using Hidden Fields](#).
 - [SCSChatDelegate](#) added in Live Agent Chat to monitor state change events, errors, and session end events.
 - [SOSStopReason](#) simplified in SOS so that it no longer has error conditions already specified by [SOSErrorCode](#).

Known Issues:

- When the same article is in multiple categories, duplicate articles can appear in the Knowledge article list and the Show More button may not work properly. **Workaround:** Don't assign multiple categories to one article.
- Some accessibility issues still need to be resolved in Knowledge: showing interface does not announce title; unable to swipe to Contact and Search icon; unable to swipe to move out of Contact and Search icons; swiping from category focus does not work correctly.
- Some accessibility issues still need to be resolved in Case Management: unable to navigate into and out of navigation bar with swipe; can't move to next field with swipe in case publisher view; unable to use swipe to get to Create or Close button in case list view; unable to navigate into and out of text fields with swipe in case detail view; unable to navigate to Send button with swipe in case detail view.
- When fetching Knowledge articles, `ViewScore` sorting does not work in this version. **Workaround:** Sort by `Title` or `LastPublishedDate` instead.
- When searching for Knowledge articles with the search bar, articles in the root category are shown in the search results even though these articles don't show from the Knowledge interface (which only shows subcategories of the root category). **Workaround:** Avoid putting essential articles in the root category; use a subcategory instead.

- (iOS 8 only) When displaying the Knowledge UI using a [custom presentation](#), the action buttons appear at the top left corner of the screen instead of horizontally aligned at the bottom.
- There is no current support for clearing the entire cache for Knowledge articles. **Workaround:** Delete the app and reinstall.

Version 1.0.0 (6/24/2016)

New Features:

- First release of a unified Service SDK for iOS that includes Knowledge, Case Management, Live Agent Chat, and SOS.
- [Offline access for Knowledge articles](#).
- [SDK customization](#) now consistent across all areas of SDK.
- SOS now provides pause functionality for both agent and customer during a session. If both parties pause, both parties need to resume for the session to continue.
- SOS now allows you to [replace UI views with your own views](#).
- [Push notifications](#) available for case feed activity.
- API enhancements.

Known Issues:

- When fetching Knowledge articles, `viewScore` sorting does not work in this version. **Workaround:** Sort by `title` or `lastPublishedDate` instead.
- In some circumstances, the Knowledge category headers overlap with the Knowledge article headers when you expand the category. We've observed this behavior on slow network connections when opening categories quickly. **Workaround:** Close and reopen the Knowledge interface.
- When searching for Knowledge articles with the search bar, articles in the root category are shown in the search results even though these articles don't show from the Knowledge interface (which only shows subcategories of the root category). **Workaround:** Avoid putting essential articles in the root category; use a subcategory instead.
- (iOS 8 only) When displaying the Knowledge UI using a [custom presentation](#), the action buttons appear at the top left corner of the screen instead of horizontally aligned at the bottom.
- There is no current support for clearing the entire cache for Knowledge articles. **Workaround:** Delete the app and reinstall.

Service Cloud Setup

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

[Cloud Setup for Knowledge](#)

Set up your knowledge base and community to use Knowledge in your app.

[Cloud Setup for Case Management](#)

Set up Case Management in Service Cloud so you can use it in your app.

[Console Setup for Live Agent Chat](#)

Set up Live Agent so you can use it in your app.

[Console Setup for SOS](#)

Set up Omni-Channel and SOS to use SOS in your app.

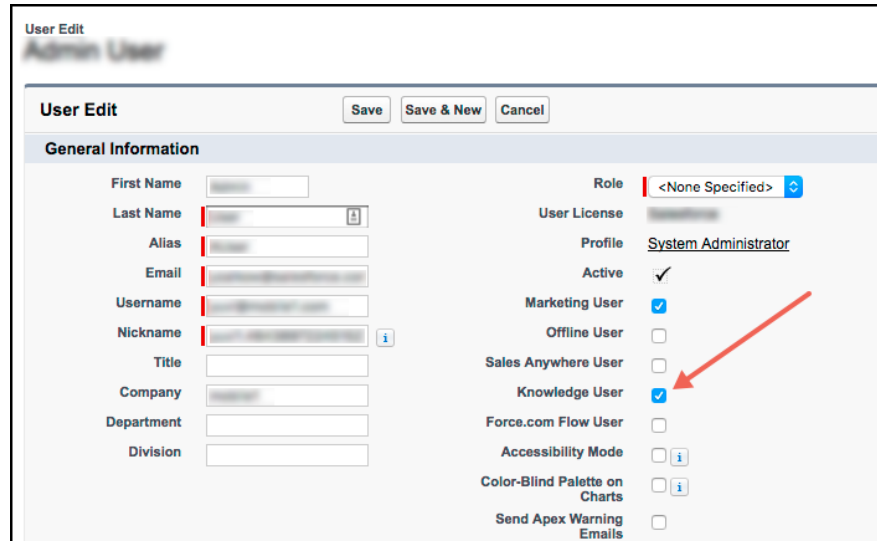
Cloud Setup for Knowledge

Set up your knowledge base and community to use Knowledge in your app.

1. **Salesforce Knowledge** must be enabled and set up in your org.

Knowledge needs to be enabled and you'll need Knowledge licenses. To learn more, see [Enable Salesforce Knowledge](#) in Salesforce Help.

Once you have Knowledge enabled, you need to turn on **Knowledge User** in the user settings for whoever administers the knowledge base.



If you don't see the **Knowledge User** checkbox, verify that you've enabled Knowledge in your org and that your org has Knowledge licenses. To learn more, see [Enable Salesforce Knowledge](#) in Salesforce Help.

2. When setting up Knowledge, make sure you define Article Types, create Knowledge articles, and associate articles with data categories within a category group.

To learn more about setting up your Knowledge articles, check out the documentation in Salesforce Help: [Salesforce Knowledge Documentation](#) ([HTML](#), [PDF](#)).

Your app developer needs the name of the **Data Category Group** and a root **Data Category** to use the knowledge feature in the SDK.

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

The screenshot shows the Salesforce Knowledge interface. The top section is titled 'Article Assignment' and includes fields for 'Assigned To', 'Assigned By', 'Instructions', and 'Assignment Due Date'. The bottom section is titled 'Article Properties' and includes fields for 'Publishing Status' (Draft), 'Type', 'Article Number' (000001001), 'Created By', and 'Last Modified By' (6/1/2016 2:49 PM). Below these are 'Categories' and 'Channels'. The 'Channels' section has four checkboxes: 'Internal App' (checked), 'Partner' (unchecked), 'Customer' (unchecked), and 'Public Knowledge Base' (checked). A red arrow points to the 'Public Knowledge Base' checkbox.

3. Your Salesforce org must have an available **Community** or **Force.com site**. Your app developer needs the **Community URL** for the site to use the Knowledge or Case Management feature in the SDK.

You can either create a site with Community Builder or you can build a Force.com site. [Getting Started With Communities](#) in Salesforce Help provides detailed information about getting a community up and running. If you don't know which type of site is suitable for your needs, check out [Choosing Between Community Builder and Force.com Sites](#).

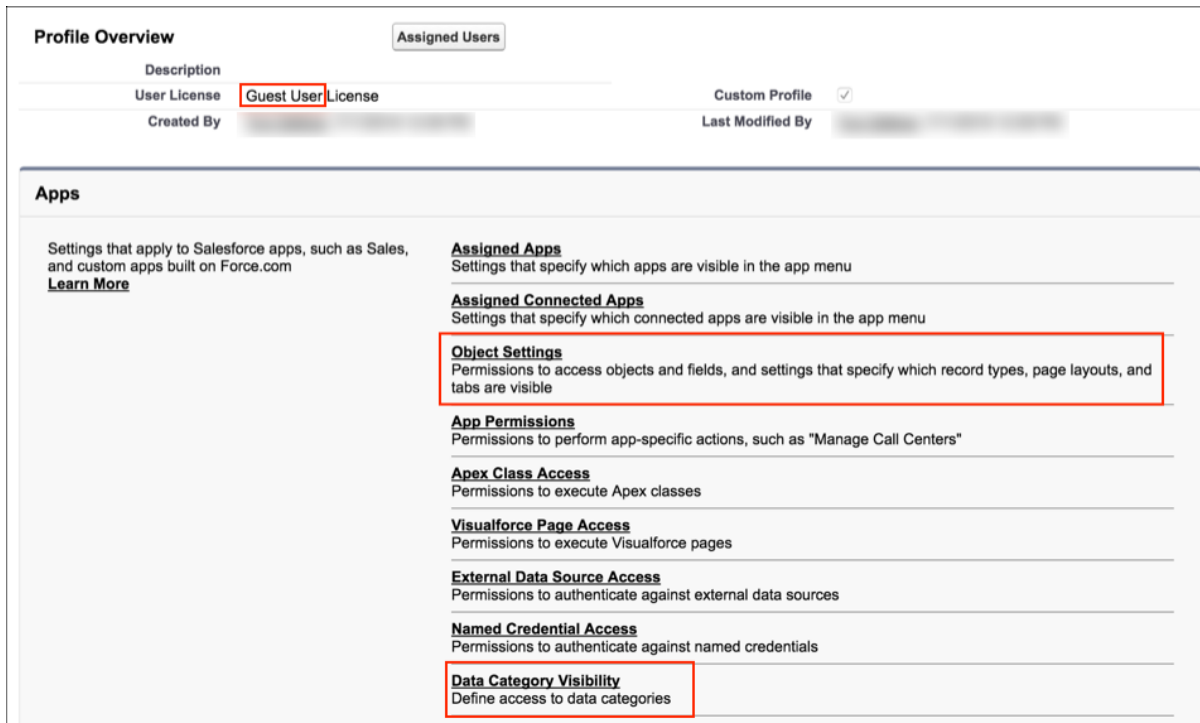
When setting up the site, be sure that your site has "Guest Access to the Support API" enabled.

The screenshot shows the 'Guest Access to the Support API' settings page. The 'Guest Access to the Support API' checkbox is checked. Below this are two sections: 'Available Quick Actions' and 'Selected Quick Actions'. The 'Selected Quick Actions' section contains a single item, 'NewCase'. There are 'Add' and 'Remove' buttons between the two sections.


4. To show Knowledge articles from your app, enable guest user access for the **Article Types**, **Categories**, and **Fields** associated with your knowledge articles.

Note: Your Knowledge categories and articles will not appear if you do not have the GUEST USER profile set up properly.

To view the guest user settings, go to your **Site Detail** page and select **Public Access Settings**. From this view, you can enable the required elements for the guest user profile. You can view the article type and article layout fields from the **Object Settings** page. You can view the categories from the **Data Category Visibility** page.



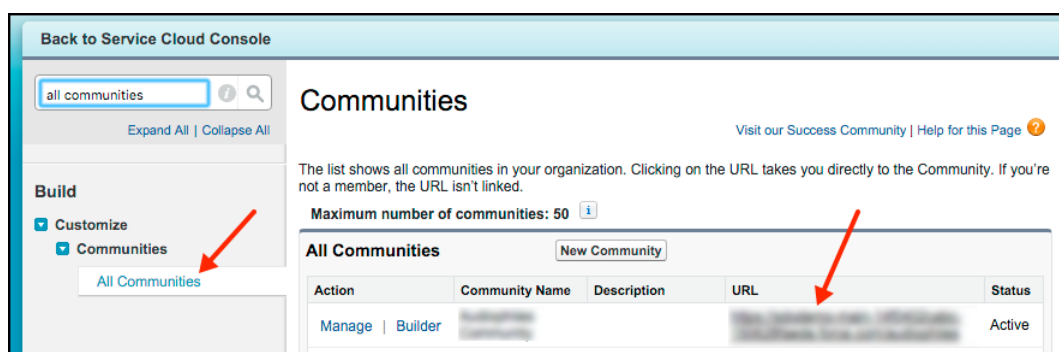
For more thorough instructions, see [Guest User Access for Your Community](#).

 **Note:** Once you've set up your knowledge base and your community, supply your app developer with three values: the community URL, the data category group, and the root data category. If you do not have this information handy, you can get it from your org's setup.

HOW TO GET THE REQUIRED VALUES FROM YOUR ORG'S SETUP:

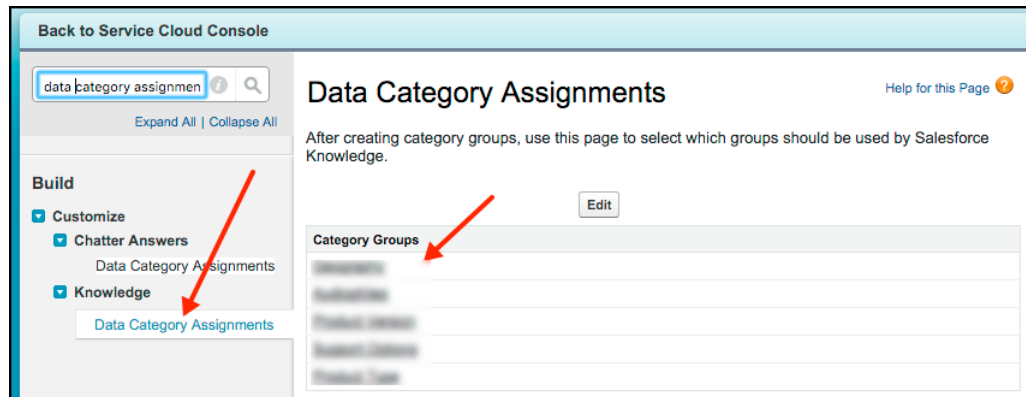
Community URL

From **Setup**, search for **All Communities**, and copy the URL for the desired community.



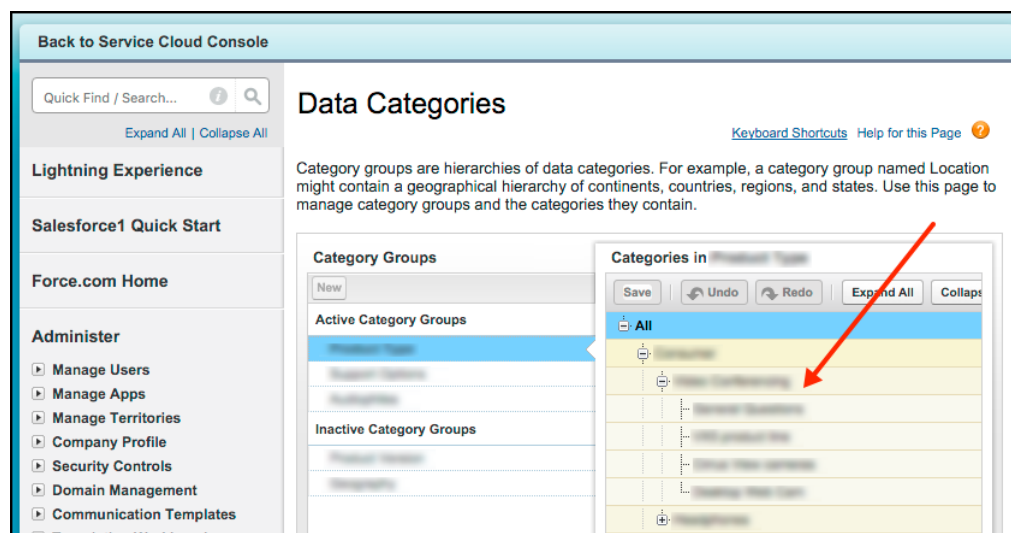
Data Category Group

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



Data Category

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



Guest User Access for Your Community

To show Knowledge articles from your app, enable guest user access for the **Article Types**, **Categories**, and **Fields** associated with your knowledge articles.

Guest User Access for Your Community

To show Knowledge articles from your app, enable guest user access for the **Article Types**, **Categories**, and **Fields** associated with your knowledge articles.

These instructions describe how to enable guest user access for either a Community or a Force.com site.

1. (Community sites only) If you are editing the settings for a **Community**:
 - a. From **Setup**, select **Customize** > **Communities** > **All Communities**.
 - b. For your chosen Community, make sure that the Status is "Active".
 - c. Click **Manage**.

- d. From the sidebar, click **Administration** > **Pages** > **Go to Force.com** to get to the **Site Detail** page.
2. (Force.com sites only) If you are editing the settings for a **Force.com site**:
 - a. From **Setup**, select **Develop** > **Sites**.
 - b. Click the **Site Label** for your site to get to the **Site Detail** page.
3. From the **Site Detail** section, click **Public Access Settings**. This action displays the guest user profile in your org.
4. Select **Data Category Visibility** and grant visibility to the categories that you want users to see.
5. Select **Object Settings** and find the Article Types that you want to be visible to users of your app.
 - a. Select **Edit** to edit the settings for each Article Type.
 - b. Make sure that **Read** is checked for the **Object Permissions** of the Article Type.
 - c. Make sure that **Read Access** is checked for the **Field Names** that you want visible.

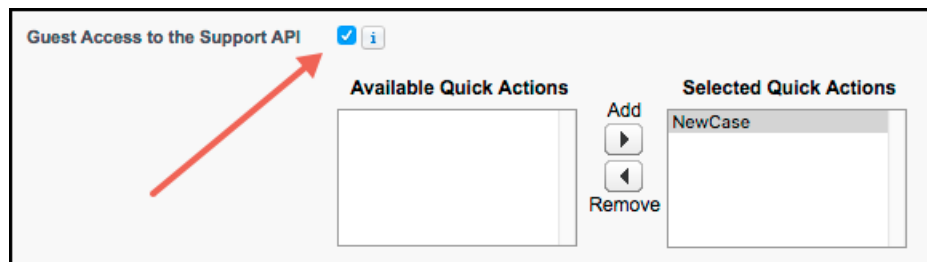
Cloud Setup for Case Management

Set up Case Management in Service Cloud so you can use it in your app.

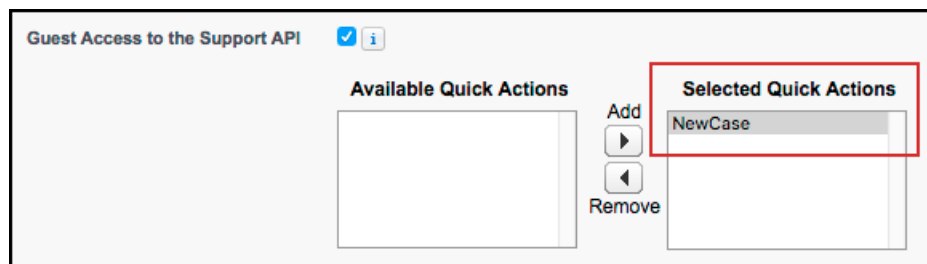
1. Your Salesforce org must have an available **Community** or **Force.com site**. Your app developer needs the **Community URL** for the site to use the Knowledge or Case Management feature in the SDK.

You can either create a site with Community Builder or you can build a Force.com site. [Getting Started With Communities](#) in Salesforce Help provides detailed information about getting a community up and running. If you don't know which type of site is suitable for your needs, check out [Choosing Between Community Builder and Force.com Sites](#).


When setting up the site, be sure that your site has "Guest Access to the Support API" enabled.



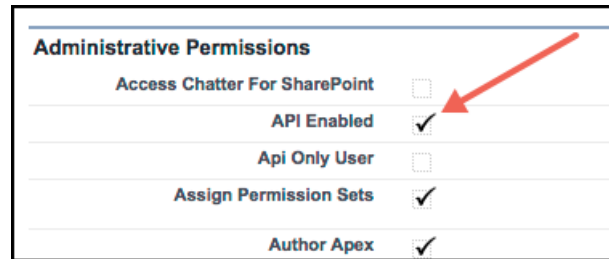
2. When setting up the site, add the **Quick Actions** that you'd like to use in your app for the Case Management functionality. You must specify a quick action to use Case Management.




The global quick action determines which fields display when creating a case. To learn more about quick actions, see [Create Global Quick Actions](#) in Salesforce Help. Supply the name of this quick action to your app developer.

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

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



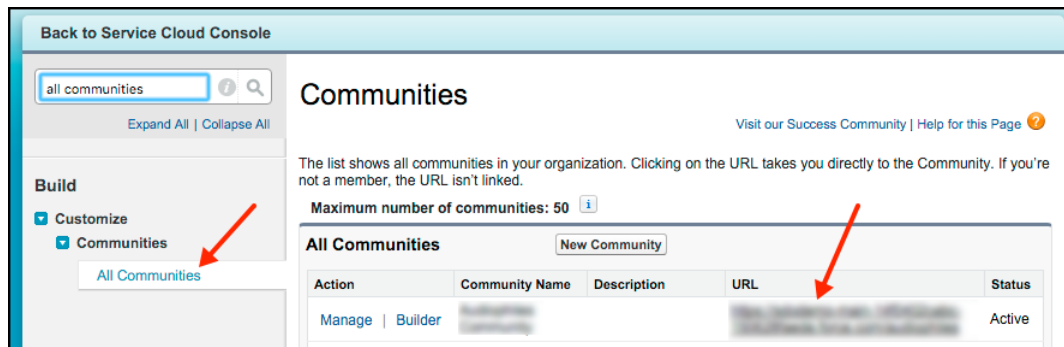
- b. You'll need a list view for your cases in Service Cloud. To learn more about creating views, see [Create a List View](#) in Salesforce Help. Supply the **Case List Unique Name** for this view to your app developer.

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

HOW TO GET THE REQUIRED VALUES FROM YOUR ORG'S SETUP:

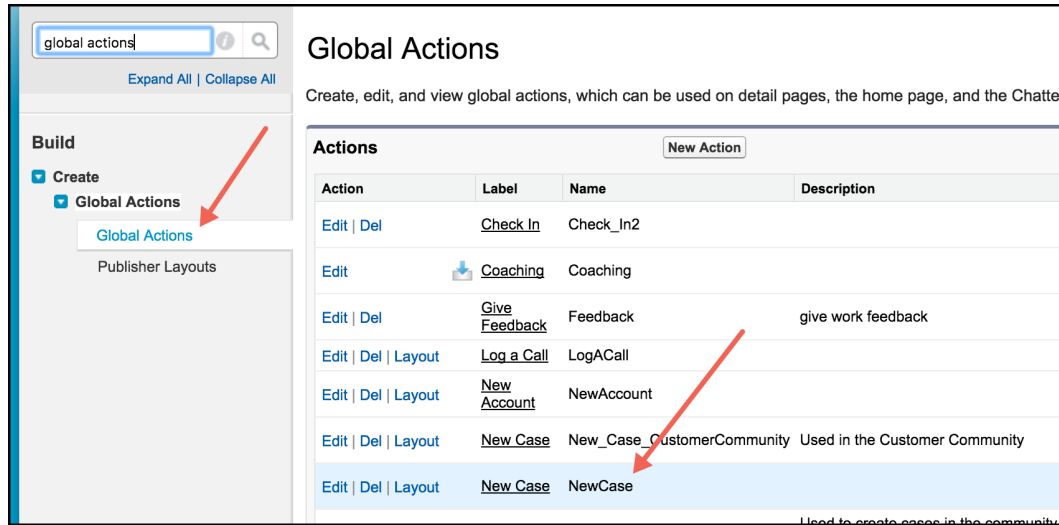
Community URL

From **Setup**, search for **All Communities**, and copy the URL for the desired community.



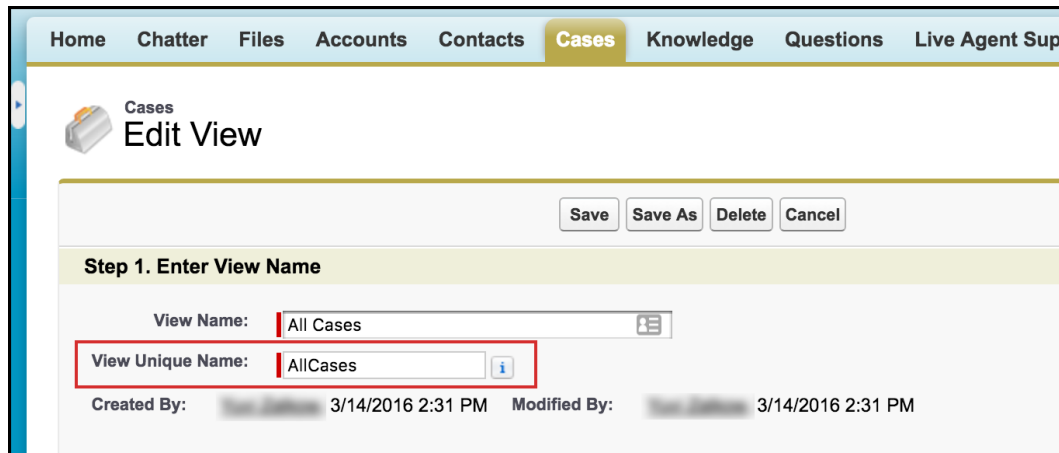
Global Action

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



Case List Unique Name

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



Console Setup for Live Agent Chat

Set up Live Agent so you can use it in your app.

1. Create a Live Agent Chat implementation in Service Cloud, as described in [Live Agent for Administrators \(PDF\)](#).
2. (Optional) If you want to use Omni-Channel, configure it as described in [Omni-Channel for Administrators \(PDF\)](#).

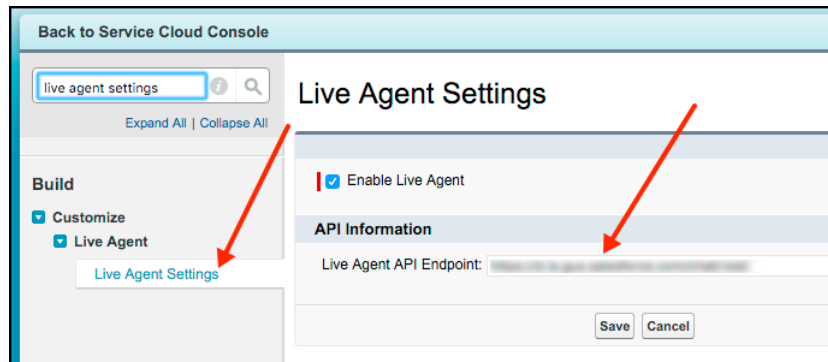
Omni-Channel allows your agents to handle all real-time routing (for example, Live Agent Chat, SOS, email, case management) with the same widget. You can still use Live Agent Chat without setting up Omni-Channel if you don't care for this feature.

Note: Once you've set up Live Agent in the console, **supply your app developer with four values:** the Live Agent pod endpoint, the organization ID, the deployment ID, and the button ID. If you do not have this information handy, you can get it from your org's setup.

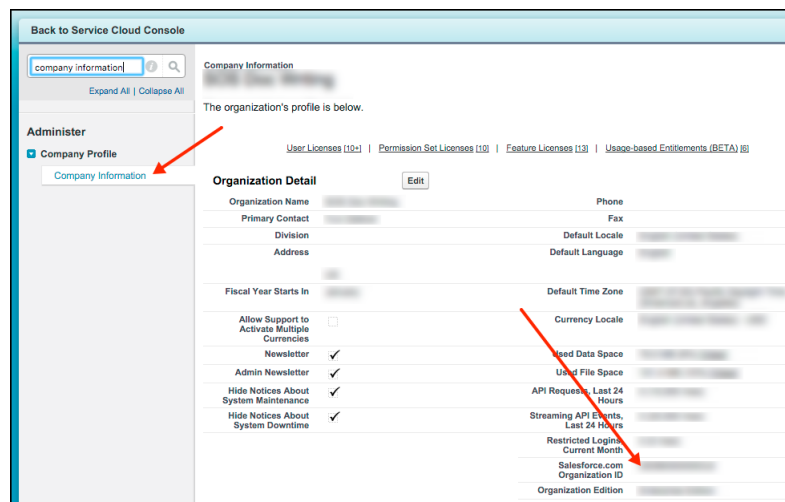
HOW TO GET THE REQUIRED VALUES FROM YOUR ORG'S SETUP:

pod

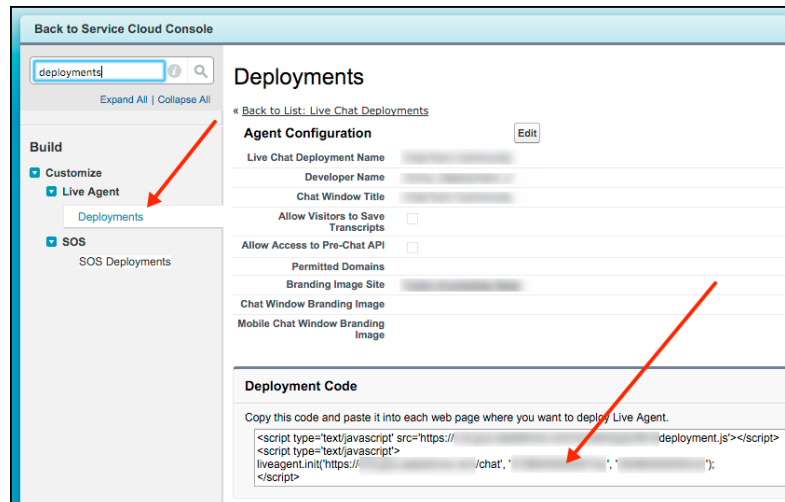
The hostname for the Live Agent pod that your organization has been assigned. To get this value, from **Setup**, search for **Live Agent Settings** and copy the hostname from the **Live Agent API Endpoint**.

**orgId**

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

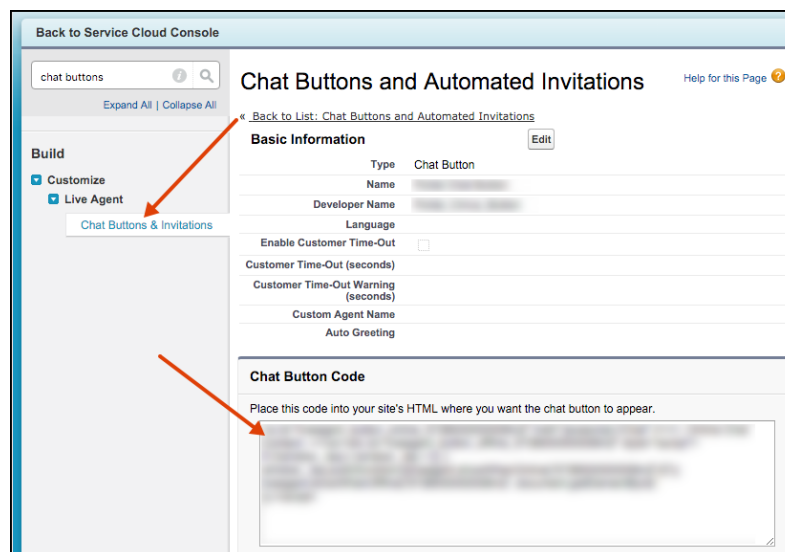
**deploymentId**

The unique ID of your Live Agent deployment. To get this value, from **Setup**, select **Live Agent > 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.



buttonId

The unique button ID for your chat configuration. You can get the button ID by creating a Live Agent chat button (see [Create Chat Buttons](#) in the Live Agent help documentation), and instead of using the supplied JavaScript, just copy the `id` attribute. To get this value after creating a button, from **Setup**, search for **Chat Buttons** and select **Chat Buttons & Invitations**. Copy the `id` for the button from the JavaScript snippet.



Console Setup for SOS

Set up Omni-Channel and SOS to use SOS in your app.

You can either use the quick setup or configure your org manually. The quick setup ([Quick Setup: SOS Console](#)) is best to get started the first time; manual setup ([Manual Setup: SOS Console](#)) is appropriate if you want to customize your org for production.

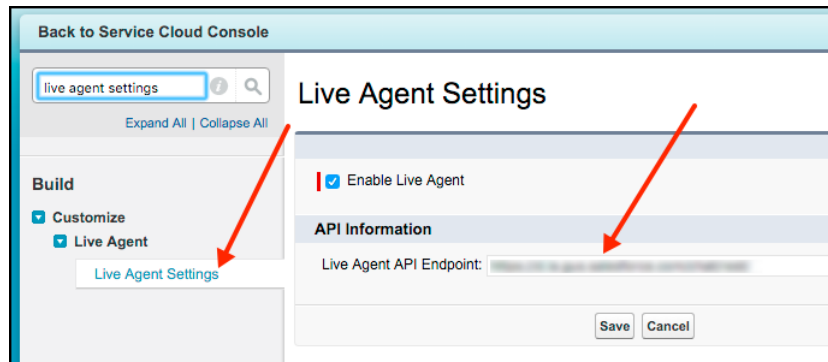


Note: Once you've set up SOS in the console, **supply your app developer with three values:** the Live Agent pod endpoint, the organization ID, and the deployment ID. If you do not have this information handy, you can get it from your org's setup.

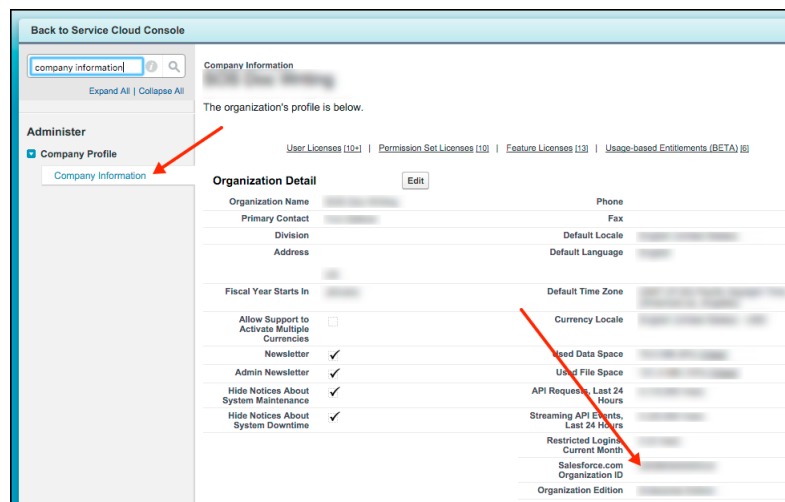
HOW TO GET THE REQUIRED VALUES FROM YOUR ORG'S SETUP:

pod

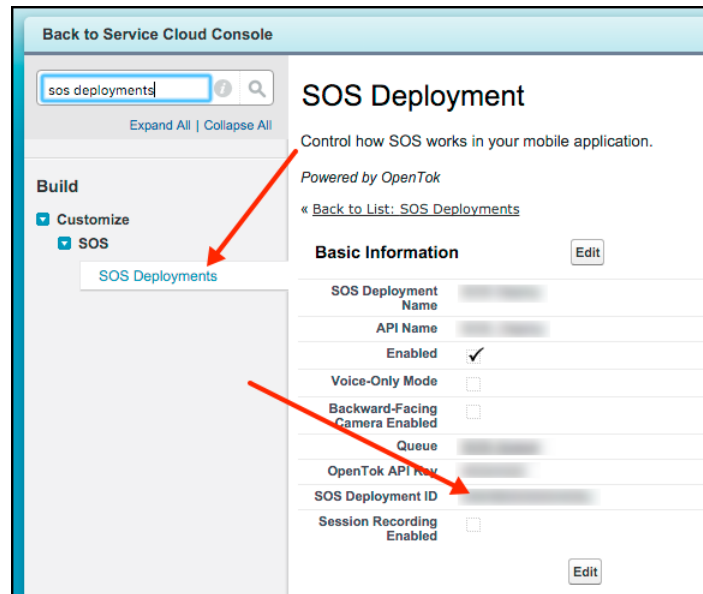
The hostname for the Live Agent pod that your organization has been assigned. To get this value, from **Setup**, search for **Live Agent Settings** and copy the hostname from the **Live Agent API Endpoint**.

**orgId**

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

**deploymentId**

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



[Quick Setup: SOS Console](#)

Quick setup is great when you want to try SOS for the first time and you haven't already enabled Omni-Channel or SOS in your org.

[Manual Setup: SOS Console](#)

Perform a manual setup when you want to fine-tune your Service Cloud for a production environment.

[Additional Setup Options: SOS Console](#)

Fine-tune your SOS configuration with additional customizations.

Quick Setup: SOS Console

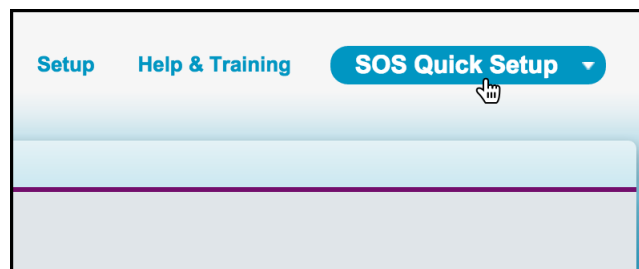
Quick setup is great when you want to try SOS for the first time and you haven't already enabled Omni-Channel or SOS in your org.

Before running through this quick start, be sure that the SOS Quick Start Package is installed into your Service Cloud instance.

- [Install Quick Start Package in your sandbox org](#)
- [Install Quick Start Package in your production org](#)

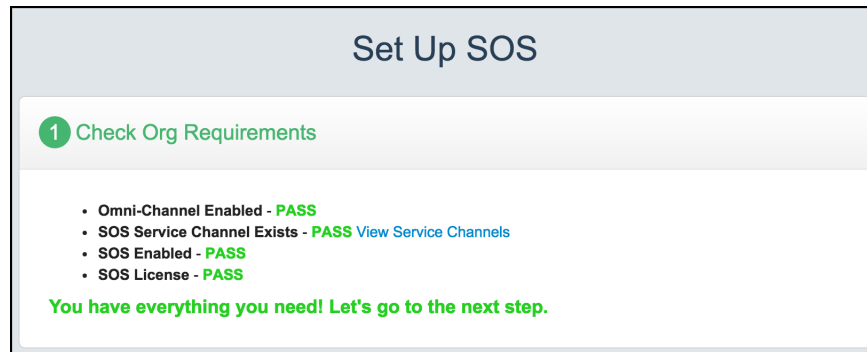
To simplify the configuration for SOS Service Cloud, we have included an easy-to-use SOS quick setup wizard.

1. Log in to your org and select the **SOS Quick Setup** app.



2. Select **Check Org Requirements**.

This step performs basic checks to ensure that your org is set up correctly. Ensure that every line item is marked with **PASS**. You can edit any settings that have not passed.



Set Up SOS

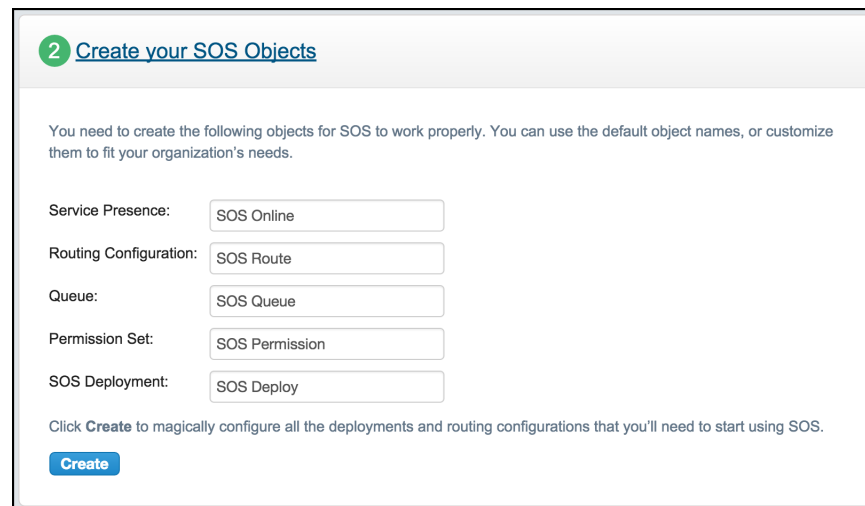
1 Check Org Requirements

- Omni-Channel Enabled - **PASS**
- SOS Service Channel Exists - **PASS** [View Service Channels](#)
- SOS Enabled - **PASS**
- SOS License - **PASS**

You have everything you need! Let's go to the next step.

3. Select **Create Your SOS Objects**.

This step allows you to provide custom names for the new SOS objects. You can leave the default values.



2 Create your SOS Objects

You need to create the following objects for SOS to work properly. You can use the default object names, or customize them to fit your organization's needs.

Service Presence:

Routing Configuration:

Queue:

Permission Set:

SOS Deployment:

Click **Create** to magically configure all the deployments and routing configurations that you'll need to start using SOS.

Create

Service Presence

Determines what appears in the Omni-Channel widget. You can edit this object and add more Service Channels to the Presence status.

Routing Configuration

Allows you to set agent work capacity and priority.

Queue

Connects the users to a Routing Configuration. Also allows you to state which objects (in this case, the SOS session) can be owned by this queue.

Permission Set

Contains the app permissions to enable the SOS license and to provide access to the Service Presence status. This object must be used to enable the license on a user. All members of the permission set must be assigned an SOS license. It can also be used to enable the Service Presence status.

SOS Deployment

Links your customer-facing application to the SOS Queue. Once created, you can configure the deployment to enable session recording by providing Amazon AWS credentials.

4. Click `Create`.

Ensure that every line item is marked with **PASS**. You can edit any settings that have not passed.

The screenshot shows a 'Create' button at the top. Below it is a 'Results' section with a bulleted list of items, each marked as 'PASS' and followed by an 'Edit' link:

- Service Presence: **PASS** - Already exists [Edit](#)
- Routing Configuration: **PASS** - Already exists [Edit](#)
- Queue: **PASS** - Already exists [Edit](#)
- Permission Set: **PASS** - Already exists [Edit](#)
- SOS Deployment: **PASS** - Updated Enabled [Edit](#)

Below the list is a paragraph of text: "If you are a Live Agent user, and would like a single 'Service Presence Status' for both SOS and Live Agent, click 'Edit' for the Service Presence above and add 'Live Agent' to the selected channels. For more info on Live Agent and the Omni-Channel you can click [here](#)".

5. Add agents to the `Set Up Users` section.

This section allows you to assign the SOS license to an agent and add the agent to the permission set and queue.

The screenshot shows the 'Set Up Users' section. It contains a paragraph of text: "For users to be able to access SOS, they must be enabled to use Service Cloud and be assigned to the correct queue and permission set. Sounds like a lot of work, doesn't it? Well, not to worry! We'll do all the heavy lifting and get your users set up for you."

Below the text are two numbered steps:

1. Enter the name of the user that you want to use SOS, or select the user's name from the drop-down list.
2. Click **Set Up User**

There are two input fields: a text box and a dropdown menu labeled "-- SELECT USER --". Both have a "Find" button and a "Clear" button next to them. Below the dropdown menu is a "Set Up User" button.

6. Select `Add the Omni-Channel Widget to Your Console App`.

Choose the **Service Cloud Console** and click **Update App**.

The screenshot shows the '3 Add the Omni-Channel Widget to your Console App' section. It contains a paragraph of text: "To use SOS, you need to enable Omni-Channel and SOS widgets in your console application."

Below the text are two numbered steps:

1. Select the console app that you want to include SOS.
2. Click **Update App**.

Below the steps is a paragraph of text: "Sit back and relax. This process might take up to a minute."

There is a dropdown menu labeled "Service Cloud Console" and an "Update App" button.

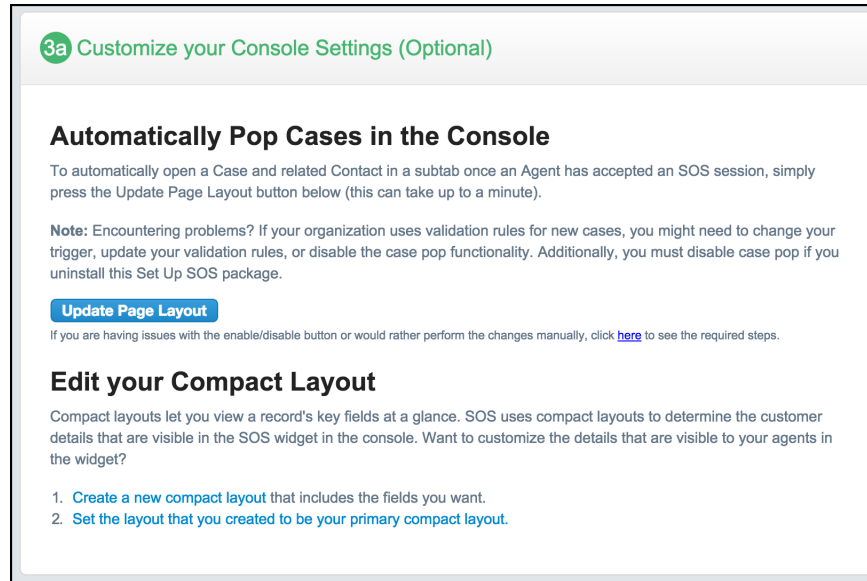
At the bottom, there is a link: "Having issues? [Add the Omni-Channel and SOS widgets to your console app manually.](#)"

This step adds the following to your Console App:

- Omni-Channel Widget
- SOS Console component
- Whitelisted domains necessary for SOS
- Report dashboard to the Navigation tab

7. (Optional) Select `Custom Your Console Settings`.

You can set your org to automatically create a case for each SOS session and open it in a subtab.

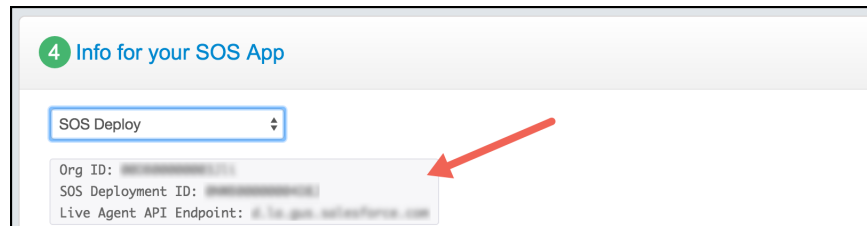


The case is created by a trigger included with the package. Before the new SOS session is inserted, the trigger creates a case and adds a reference to it to the SOS session object. If you wish to modify the trigger, it can be found in **Setup**, by searching **SOS Sessions** and going to **Triggers**. The name of the trigger is **SOSCreateCase**.

The case is popped in a subtab by a page that is hidden in the SOS session page layout. This hidden page and an altered SOS session page layout are also included with the package.

8. Select **Info for Your SOS App**.

This step provides you with the three pieces of information required to start an SOS session from the SDK: Organization ID, SOS Deployment ID, and Live Agent API Endpoint. Save this information for later.



Once you've completed these steps, you are ready to start using the SOS feature in the Service SDK.

Manual Setup: SOS Console

Perform a manual setup when you want to fine-tune your Service Cloud for a production environment.

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

Additional Setup Options: SOS Console

Fine-tune your SOS configuration with additional customizations.

[Assign SOS Permissions](#)

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

[Automatic Case Pop](#)

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

[Record SOS Sessions](#)

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

[SOS Reference ID](#)

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

[Multiple Queues](#)

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

Assign SOS Permissions

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

1. Assign an SOS user license.

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

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

2. Enable the SOS license.

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

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

3. Enable the service presence status.

You can enable the service presence status using either permission sets or profiles. If the presence status is only being used for SOS, it is easier to enable the presence status through the same permission set that enables the license. Using the same permission set guarantees that all agents who require the presence status have access to it. If the presence status is being used for multiple service channels, it is likely that the same permission set cannot be used, since all members of the permission set would require a SOS

license. In this case, you may want to have multiple permissions sets, assign it to a profile, and use some combination of profiles and permission sets.

Service Permission via Permission Sets

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

Service Permission via Profile

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

4. Add agents to the queue.

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

- a. From **Setup**, select **Manage Users > Queues**.
- b. Click **Edit** for the desired queue.
- c. Scroll to the bottom of the page and find the **Queue Members** section. Add the required members.
- d. Click **Save**.

Automatic Case Pop

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

1. Create a trigger.

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

- a. From **Setup**, search for **SOS Sessions**.
- b. Select **Triggers** from the **SOS Sessions** section.
- c. Click the **New** button.

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


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

- e. Click **Save**.

2. Add a Visualforce page.

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

- From **Setup**, search for **Visualforce Pages**.
- Click the **New** button.
- Give the Visualforce page a name. For example, "SOS_Open_Case_Custom".

 **Note:** The **Label** field can contain spaces but the **Name** field cannot.

- d. Replace the **Visualforce Markup** code with this code:

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

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

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

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

```

        return;
    }

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

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

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

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

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

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

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

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

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

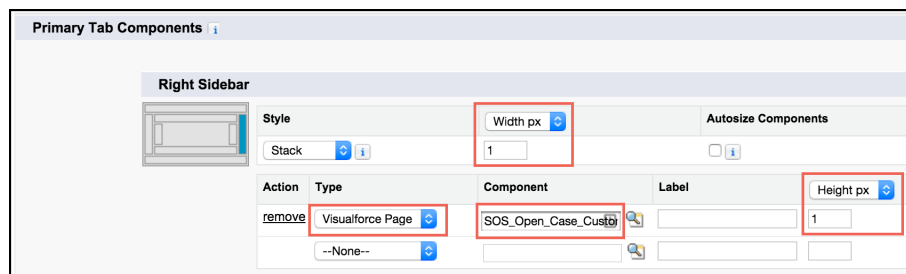
```

e. Click **Save**.

3. Update the SOS session page layout.

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

- a. From **Setup**, search for **SOS Sessions**.
- b. Select **Page Layouts** from the **SOS Sessions** section.
- c. Click **Edit** for your active layout (probably **SOS Session Layout**).
- d. From the top of the page, select the **Custom Console Components** link.
- e. Under the **Primary Tab Components** section, add the following to one of the sidebars:



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

Record SOS Sessions

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

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

You can retrieve recorded sessions in the [mp4](#) format from your Amazon S3 bucket.

SOS Reference ID

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

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

There are two ways to find the SOS Reference ID:

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

Add to Session Object

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

1. From **Setup**, search for **SOS Sessions**.
2. From **SOS Sessions**, select **Fields**. (Do not go to Fields under SOS Session Activities.)
3. Click **New** under **SOS Session Custom Fields & Relationships**.
4. Select **Formula**. Click **Next**.
5. Enter **SOS Reference Id** as the **Field Label**. **Field Name** auto populates.
6. Select **Text** as the **Formula Return Type**. Click **Next**.
7. In the **Simple Formula** text area, enter **Id**. Click **Next**.
8. Click **Next** again. (Permission to view the field can be removed on this page before clicking next.)
9. Click **Save**.

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

Add to Session List View

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

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

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

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

Multiple Queues

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

Multiple queues can help out in the following situations:

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

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

1. If a `Routing Configuration` is being used to achieve different priorities, create this object first. If you want all queues to have the same priority, the same routing configuration can be used.
2. Next, create the `Queue`. The queue references the routing config. An agent can be a member of multiple queues.
3. Create the `SOS Deployment` last. The deployment references the queue. An app may have access to several SOS deployment IDs, and then the app decides which queue the user should be sent to using the SOS deployment ID.

SDK Setup

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

[Requirements](#)

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

[Install the SDK](#)

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

[Prepare Your App for Submission](#)

Before you can submit your app to the App Store, you need to strip development resources (such as unneeded architectures and header resources) from the dynamic libraries used by the Service SDK.

Requirements

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

SDK Development Requirements

To develop using this SDK, you must have:

- [iOS SDK](#) version 8 or newer
- [Xcode](#) version 8 or newer

App Requirements

Any app that uses this SDK requires:

- [iOS](#) version 8 or newer

SOS Agent Requirements

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

Hardware requirements:

- Webcam
- Microphone

Bandwidth requirements:

- 500 Kbps upstream
- 500 Kbps downstream

Browser requirements:

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

Operating System:

- OSX 10.5 or newer
- Windows 7 or newer

Install the SDK

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

1. Add the SDK frameworks.

You can add the frameworks manually or add them using [CocoaPods](#). CocoaPods is a popular dependency manager for Swift and Objective-C projects.

- [Add the Frameworks with CocoaPods](#)
- [Add the Frameworks Manually](#)

2. (Knowledge only) Add an [App Transport Security \(ATS\)](#) exception to `localhost` for serving cached knowledge base articles.

a. Open the `Info.plist` file for your project.



Note: If you right-click the `plist` file from the project navigator, you can select **Open As > Source Code** from the context menu. The source code view is a quick way to add domains to your `plist` file.

b. Add `NSAppTransportSecurity` to your `Info.plist` to allow insecure HTTP loads from `localhost`.

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSExceptionDomains</key>
  <dict>
    <key>localhost</key>
    <dict>
      <key>NSExceptionAllowsInsecureHTTPLoads</key>
      <true/>
    </dict>
  </dict>
</dict>
```

3. (SOS only) If you're using SOS, iOS 10 requires descriptions for why the app needs to access the device's microphone and camera.

Add string values for `NSMicrophoneUsageDescription` and `NSCameraUsageDescription` 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>Camera is used for SOS video chat with an agent.</string>
<key>NSMicrophoneUsageDescription</key>
<string>Microphone is used for SOS chat with an agent.</string>
```

4. (SOS only) If you're using SOS, turn on **Background Modes** for your project.

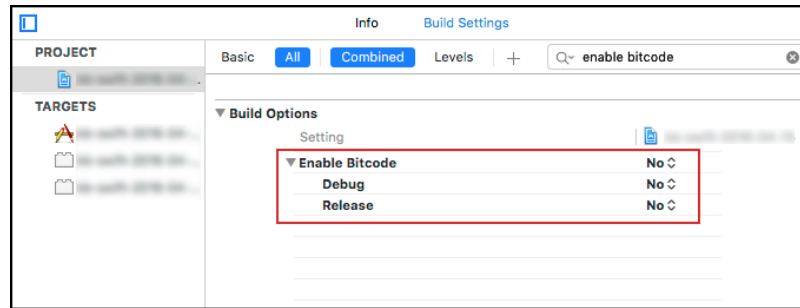
To ensure that an SOS and Live Agent Chat session remains active while the app is in the background, verify that your background settings are correct.

- Open your Project Target Settings.
- Select the **Capabilities** tab.
- Set the **Background Modes** to **ON**.
- From the **Background Modes** subcategories, check the **Audio, AirPlay, and Picture in Picture** item.

5. (SOS only) If you're using SOS, turn off **Enable Bitcode** for your project.

The OpenTok library used by SOS does not currently support [App Thinning](#) of binaries. To ensure that the SDK runs properly, Bitcode must be turned off for projects using SOS.

- a. Open your Project Target Settings.
- b. Select the **Build Settings** tab.
- c. Search for **Enable BitCode** in the provided search bar.
- d. Set the value to **NO**.



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

[Add the Frameworks with CocoaPods](#)

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

[Add the Frameworks Manually](#)


Add the SDK frameworks by manually embedding the appropriate frameworks.

Add the Frameworks with CocoaPods

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

1. If you haven't already done so, install the [CocoaPods](#) gem and initialize the CocoaPods master 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. Add the Service Cloud SDK dependency.

- a. Change directories into the root directory of your application project and create or edit the file named `Podfile`
- b. If you want to install the Service 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 Service SDK (with all components)
target '<your app target>' do
```

```
    pod 'ServiceSDK'
end
```

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

Feature	Pod name
Knowledge	ServiceSDK/Knowledge
Case Management	ServiceSDK/Cases
Live Agent Chat	ServiceSDK/Chat
SOS	ServiceSDK/SOS

For example, the following `Podfile` installs SOS:

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

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

If you don't specify a version number, you'll 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 SDK and *not* the version number of the individual component.

For instance, if you want version 3.2.2 of SOS, specify 2.0.2 because version 2.0.2 of the Service SDK has version 3.2.2 of SOS.

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

# To use SOS (with version info)
target '<your app target>' do
  pod 'ServiceSDK/SOS', '2.0.2'
end
```

4. Run the CocoaPods installer.

```
pod install
```

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

5. 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 29.

Add the Frameworks Manually

Add the SDK frameworks by manually embedding the appropriate frameworks.

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

The following frameworks are available to you:

Framework	Description	Required?
<code>SalesforceKit</code>	This framework gives you access to the Salesforce Mobile SDK. See the Mobile SDK Developer's Guide for more info.	Yes
<code>ServiceCore</code>	Contains all the common components used by the Service SDK.	Yes
<code>ServiceKnowledge</code>	Contains access to the Knowledge features of the SDK.	Only if using Knowledge
<code>ServiceCases</code>	Contains access to the Case Management features of the SDK.	Only if using Case Management
<code>ServiceChat</code>	Contains access to the Live Agent Chat features of the SDK.	Only if using Live Agent Chat
<code>ServiceSOS</code>	Contains access to the SOS features of the SDK.	Only if using SOS

Add the relevant frameworks to the **Embedded Binaries** section of the **General** tab for your target app. Be sure to select **Copy items if needed** when embedding.

Once you've embedded the frameworks, you'll automatically see them appear in the **Linked Frameworks and Libraries** section as well. If you see two line items for each framework (which happens if you drag the frameworks into the project before embedding), delete the duplicates.

Once you've added the SDK frameworks, proceed with [the installation instructions](#) on page 29.

Prepare Your App for Submission

Before you can submit your app to the App Store, you need to strip development resources (such as unneeded architectures and header resources) from the dynamic libraries used by the Service SDK.

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

For example, if the framework is in your main project directory, use:

```
"$SRCROOT/SalesforceKit.framework/prepare-framework"
```

And if you've installed the SDK with CocoaPods, use:

```
"$PODS_ROOT/ServiceSDK/Frameworks/SalesforceKit.framework/prepare-framework"
```

 **Note:** This build phase must occur *after* the link and embed phases.

Quick Start Tutorials

Get going quickly with these short introductory tutorials.

[Get Started with Knowledge](#)

It's easy to wire up your iOS app to your knowledge base articles.

[Get Started with Case Publisher](#)

Quickly build an app that lets you create a new case.

[Get Started with Live Agent Chat](#)

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

[Get Started with SOS](#)

See for yourself how easy and effective live video chat and screen sharing can be.

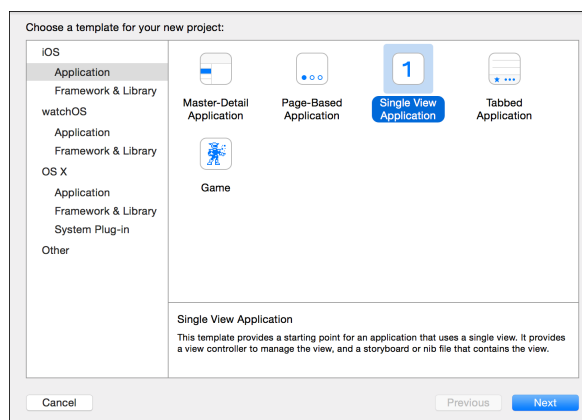
Get Started with Knowledge

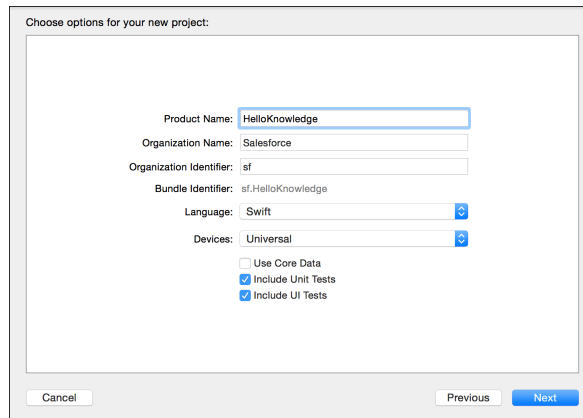
It's easy to wire up your iOS app to your knowledge base articles.

Before doing this tutorial, be sure that you've set up Service Cloud for Knowledge. See [Cloud Setup for Knowledge](#) for more information.

This tutorial shows you how to put a knowledge base into your iOS app.

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





2. Install the SDK as described in [Install the SDK](#).
3. From your app delegate implementation, import the SDK.

In Objective-C:

```
@import ServiceCore;
#import ServiceKnowledge;
```

In Swift:

```
import ServiceCore
import ServiceKnowledge
```

4. Point the SDK to your org from the `applicationDidFinishLaunchingWithOptions` method of your app delegate implementation.

To connect your app to your organization, create an [SCSServiceConfiguration](#) object containing the community URL, the data category group, and the root data category. Pass this object to the [SCServiceCloud](#) shared instance.

In Objective-C:

```
// Create configuration object with init params
SCSServiceConfiguration *config = [[SCSServiceConfiguration alloc]
initWithCommunity:[NSURL URLWithString:@"https://mycommunity.example.com"]
dataCategoryGroup:@"Regions"
rootDataCategory:@"All"];

// Perform any additional configuration here


// Pass configuration to shared instance
[SCServiceCloud sharedInstance].serviceConfiguration = config;
```

In Swift:

```
// Create configuration object with init params
let config = SCSServiceConfiguration(
    community: URL(string: "https://mycommunity.example.com")!,
    dataCategoryGroup: "Regions",
    rootDataCategory: "All")

// Perform any additional configuration here
```

```
// Pass configuration to shared instance
SCServiceCloud.sharedInstance().serviceConfiguration = config
```

 **Note:** You can get the required parameters for this method from your Salesforce org. If your Salesforce Admin hasn't already set up Knowledge in Service Cloud or you need more guidance, see [Cloud Setup for Knowledge](#).

5. Enable Knowledge within the `applicationDidFinishLaunchingWithOptions` method.

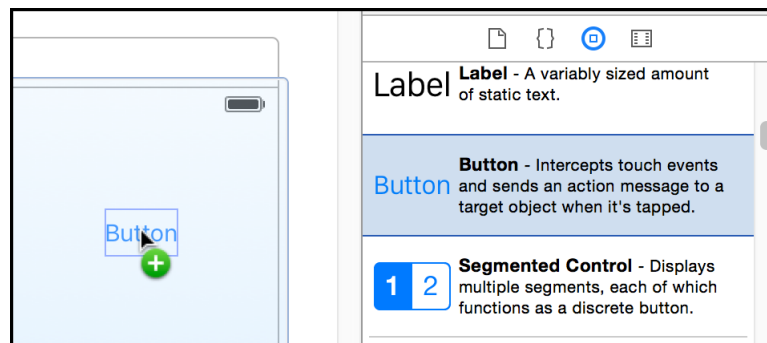
In Objective-C:

```
[SCServiceCloud sharedInstance].knowledge.enabled = YES;
```

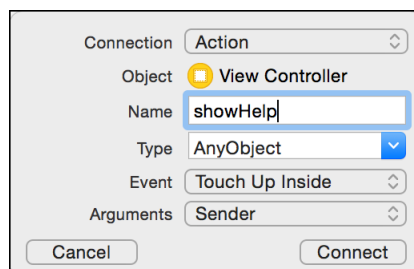
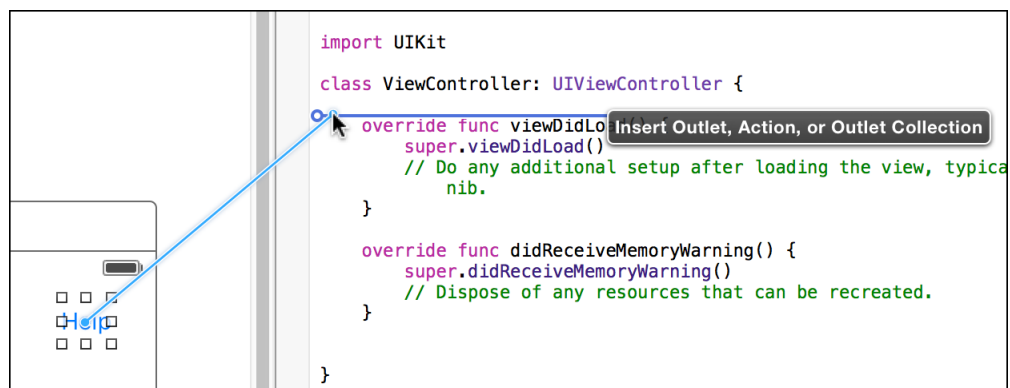
In Swift:

```
SCServiceCloud.sharedInstance().knowledge.enabled = true
```

6. Go to your storyboard and place a button somewhere on the view. Name it `Help`.



7. Add a `Touch Up Inside` action to your `UIViewController` implementation. Name it `showHelp`.



8. From your view controller implementation, import the SDK.

In Objective-C:

```
@import ServiceCore;  
@import ServiceKnowledge;
```

In Swift:

```
import ServiceCore  
import ServiceKnowledge
```

9. From within the button action handler, activate the Knowledge interface using the `setInterfaceVisible` method.

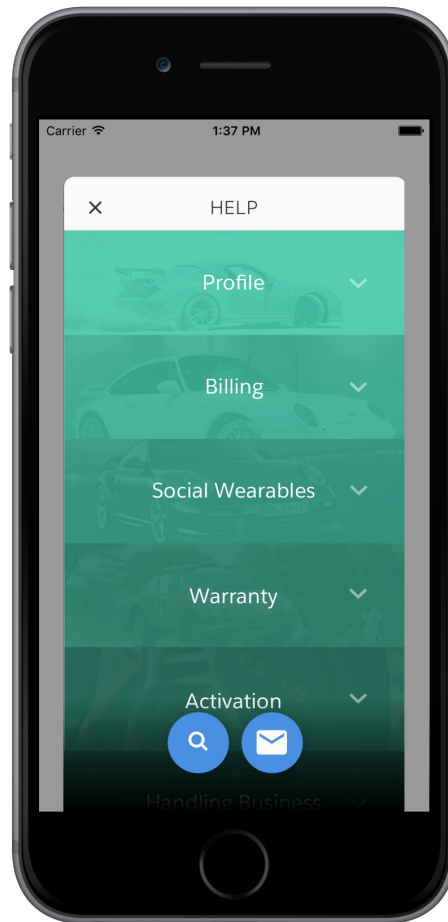
In Objective-C:

```
[[SCServiceCloud sharedInstance].knowledge setInterfaceVisible:YES  
                                           animated:YES  
                                           completion:nil];
```

In Swift:

```
SCServiceCloud.sharedInstance().knowledge.setInterfaceVisible(true,  
                                                               animated: true,  
                                                               completion: nil)
```

And that's it! You can now build and run your app to see how it looks. Click the `Help` button to activate the interface.



You can customize the interface so it looks and feels just like your app. Check out [SDK Customizations](#) for guidance in this area.

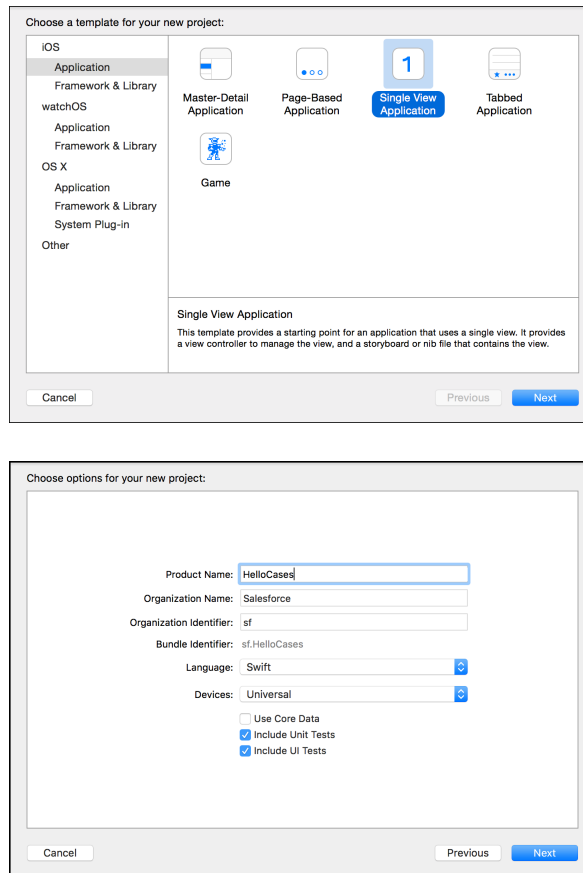
Get Started with Case Publisher

Quickly build an app that lets you create a new case.

Before doing this tutorial, be sure that you've set up Service Cloud for Case Management. See [Cloud Setup for Case Management](#) for more information.

This tutorial shows you how to connect your iOS app to the case management interface as a guest user. A guest user is able to publish new cases. However, if you'd like to manage existing cases, you'll need to authenticate a user from within your app. Authentication is discussed in [Case Management as an Authenticated User](#).

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



2. Install the SDK as described in [Install the SDK](#).
3. From your app delegate implementation, import the SDK.

In Objective-C:

```
@import ServiceCore;
#import ServiceCases;
```

In Swift:

```
import ServiceCore
import ServiceCases
```

4. Point the SDK to your org from the `applicationDidFinishLaunchingWithOptions` method in your `UIApplicationDelegate` class.

To connect your application to your organization, create an [SCSServiceConfiguration](#) object containing the community URL and pass this object to the [SCServiceCloud](#) shared instance.

In Objective-C:

```
// Create configuration object with your community URL
SCSServiceConfiguration *config = [[SCSServiceConfiguration alloc]
initWithCommunity:[NSURL URLWithString:@"https://mycommunity.example.com"]];

// Pass configuration to shared instance
[SCServiceCloud sharedInstance].serviceConfiguration = config;
```

In Swift:

```
// Create configuration object with your community URL
let config = SCSServiceConfiguration(
    community: URL(string: "https://mycommunity.example.com")!)

// Pass configuration to shared instance
SCServiceCloud.sharedInstance().serviceConfiguration = config
```

You can get the community URL from your Salesforce org. From **Setup**, search for **All Communities**, and copy the URL for the desired community. For more help, see [Cloud Setup for Case Management](#).



Note: If you plan to access Knowledge in addition to Case Management, use the `initWithCommunity:dataCategoryGroup:rootDataCategory:method` instead. This method sets up data categories in addition to setting the community URL. See [Quick Setup: Knowledge](#) in the Knowledge section for more info.

5. Enable Case Management within the `applicationDidFinishLaunchingWithOptions` method.

In Objective-C:

```
[SCServiceCloud sharedInstance].cases.enabled = YES;
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.isEnabled = true
```

6. Assign a global action to the Case Management interface. The global action determines the fields shown when a user creates a case. To configure the fields shown when creating a case, specify the global action name in the `caseCreateActionName` property. This code snippet illustrates how to associate the case publisher feature with the **New Case** global action layout, which is one of the default actions provided in most orgs.

In Objective-C:

```
[SCServiceCloud sharedInstance].cases.caseCreateActionName = @"NewCase";
```

In Swift:

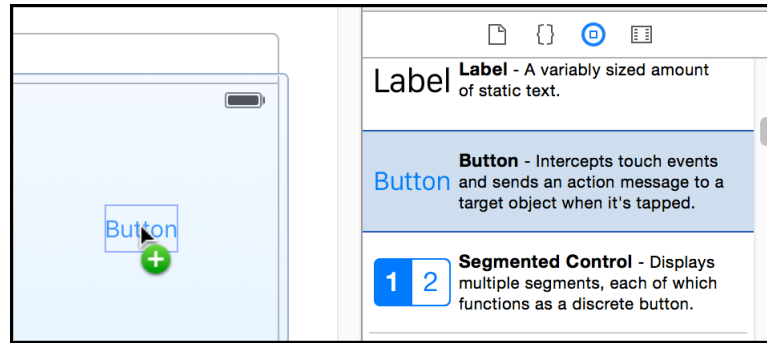
```
SCServiceCloud.sharedInstance().cases.caseCreateActionName = "NewCase"
```

You can get the global action name from your Salesforce org. From **Setup**, search for **Global Actions**, and copy the name of the desired quick action. For more help, see [Cloud Setup for Case Management](#).

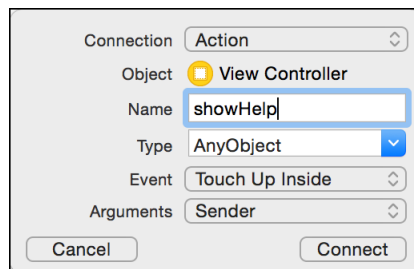
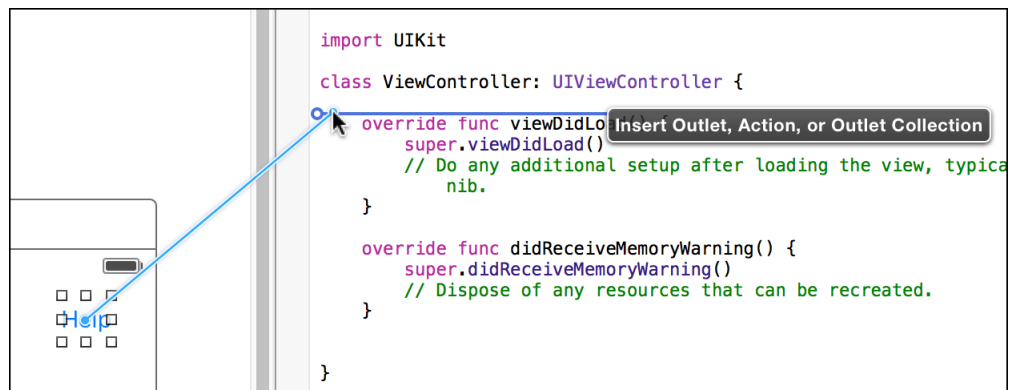


Note: Be sure that your global action is accessible to the Guest User profile. Also note that the case publisher screen does not respect field-level security for guest users. If you want to specify different security levels for different users, use different quick actions.

7. Go to your storyboard and place a button somewhere on the view. Name it `Help`.



8. Add a Touch Up Inside action to your UIViewController implementation. Name it `showHelp`.



9. From your view controller implementation, import the SDK.

In Objective-C:

```
@import ServiceCore;
#import ServiceCases;
```

In Swift:

```
import ServiceCore
import ServiceCases
```

10. From within the button action handler, activate the Case Management interface using the `setInterfaceVisible` method.

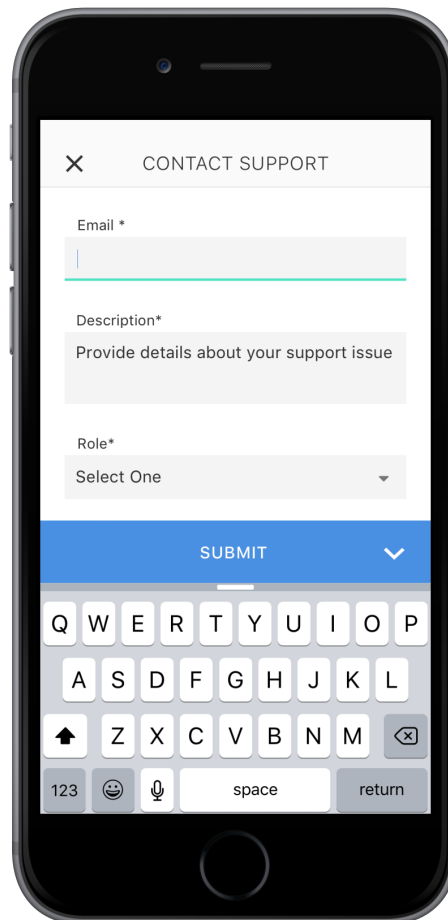
In Objective-C:

```
[[SCServiceCloud sharedInstance].cases setInterfaceVisible:YES  
                                     animated:YES  
                                     completion:nil];
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.setInterfaceVisible(true,  
                                                         animated: true,  
                                                         completion: nil)
```

And that's it! You can now build and run your app to see how it looks. Click the `Help` button to activate the interface.



If you would like to give your users access to their existing case list, you'll need to authenticate the user first. To learn more about authentication, see [Case Management as an Authenticated User](#). You can also customize the look and feel of the interface, as described in [SDK Customizations](#).

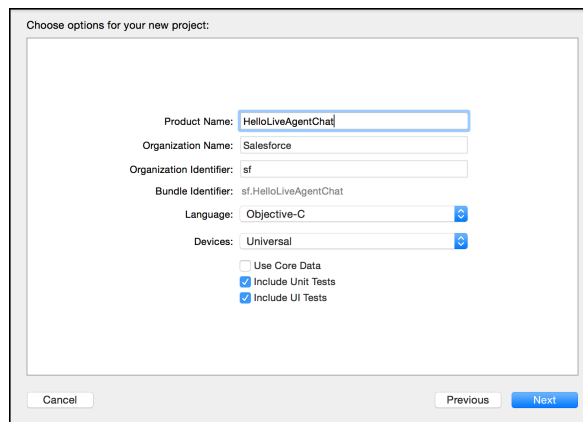
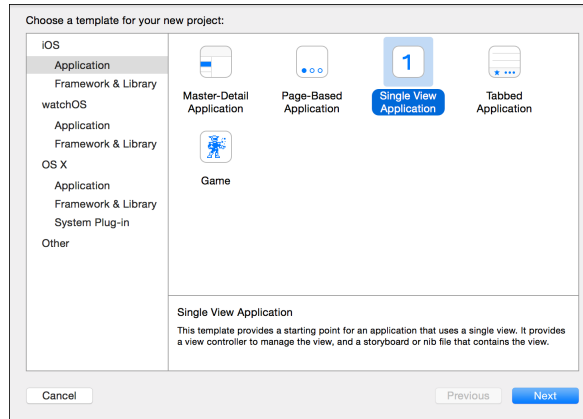
Get Started with Live Agent Chat

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

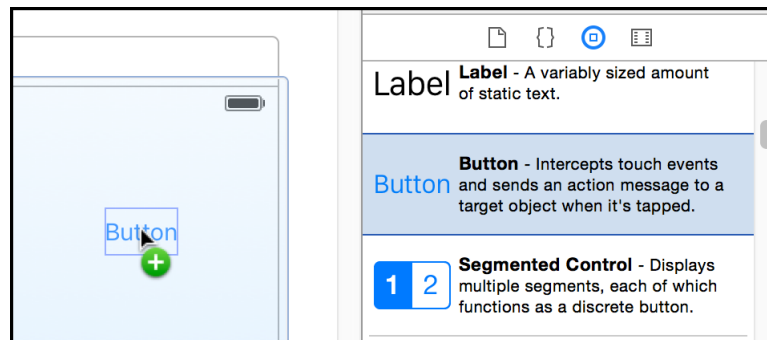
Before doing this tutorial, be sure that you've set up Service Cloud for Live Agent. See [Console Setup for Live Agent Chat](#) for more information.

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

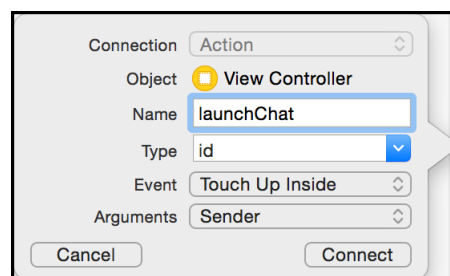
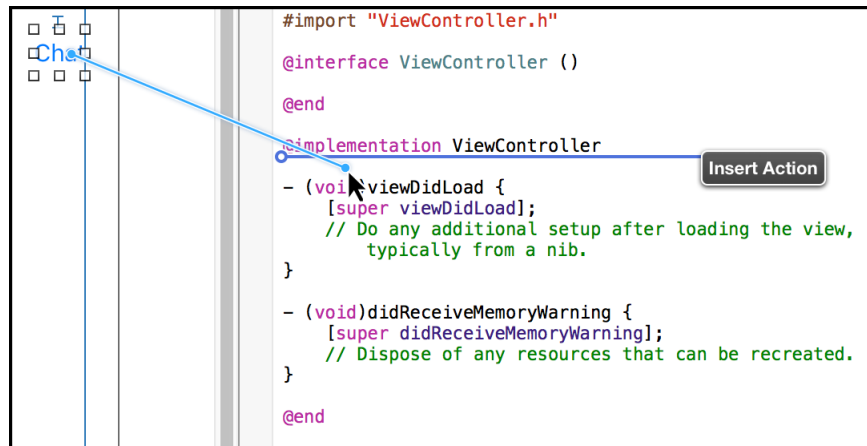
1. Create an Xcode project. For this example, let's make a Single View Application. Name it `HelloLiveAgentChat`.



2. Install the SDK as described in [Install the SDK](#).
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`.



- Import the SDK. Wherever you intend to use the Live Agent Chat SDK, be sure to import the Service Common framework and the Live Agent Chat framework.

In Objective-C:

```

#import ServiceCore;
#import ServiceChat;

```

In Swift:

```

import ServiceCore
import ServiceChat

```

- Launch a Live Agent Chat session from within the `launchChat` method.

From the button action implementation, launch Live Agent Chat using the `startSessionWithOptions` method.

In Objective-C:

```

- (IBAction)launchChat:(id)sender {

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

    // Start the session.

    [[SCServiceCloud sharedInstance].chat startSessionWithConfiguration:config];
}

```

In Swift:

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

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

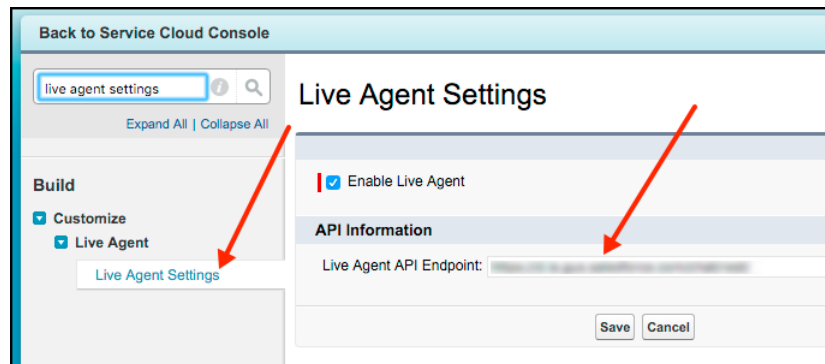
    // Start the session.

    SCServiceCloud.sharedInstance().chat.startSessionWithConfiguration(config)
}
```

Fill in the placeholder text for the Live Agent pod, the org ID, the deployment ID, and the button ID.

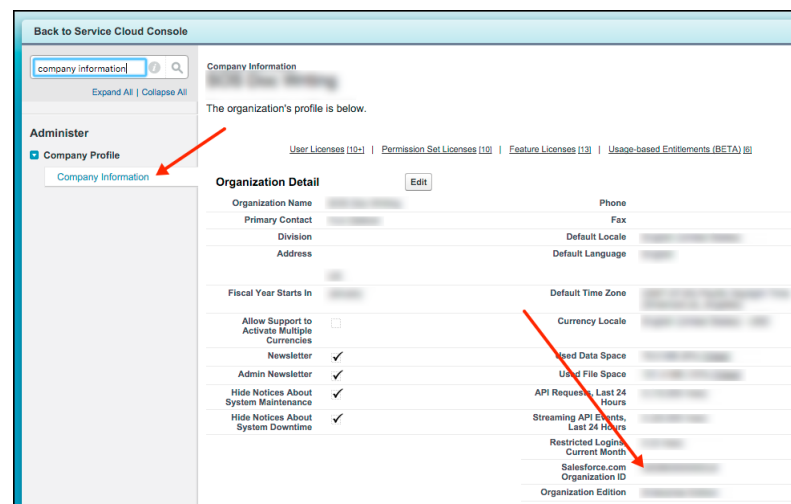
pod

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



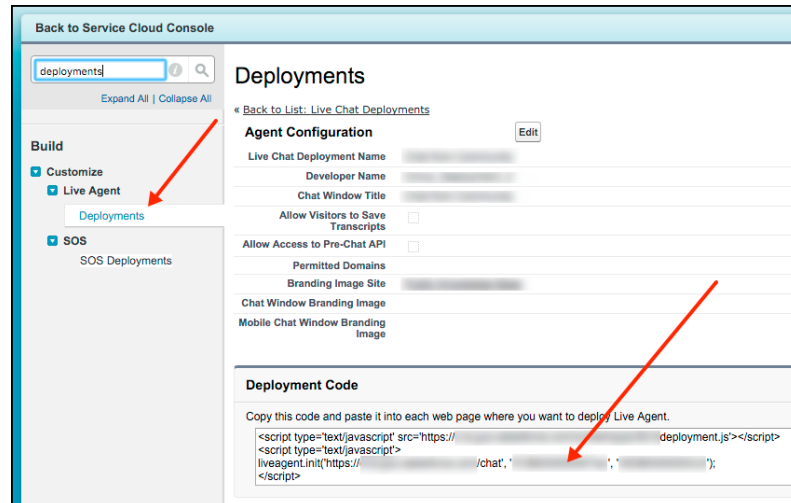
orgId

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

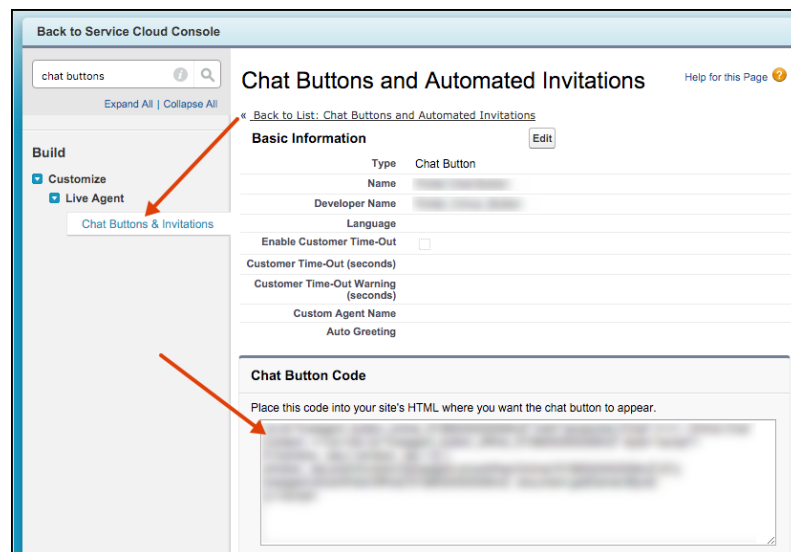


deploymentId

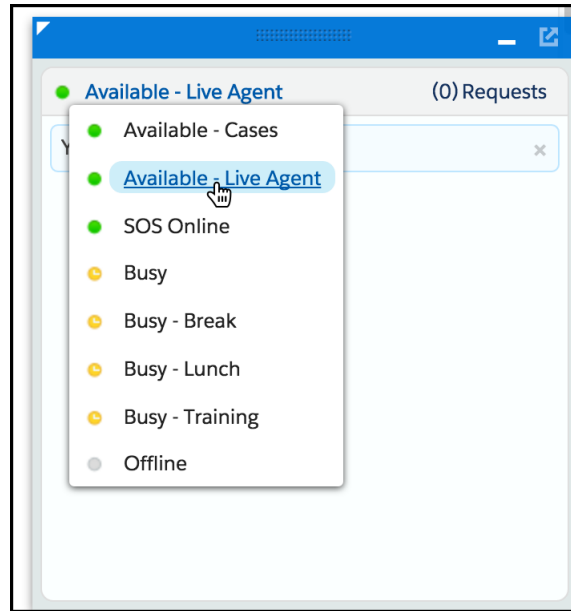
The unique ID of your Live Agent deployment. To get this value, from **Setup**, select **Live Agent > Deployments**. The script at the bottom of the page contains a call to the `liveagent.init` function with the `pod`, the `deploymentId`, and `orgId` as arguments. Copy the `deploymentId` value.

**buttonId**

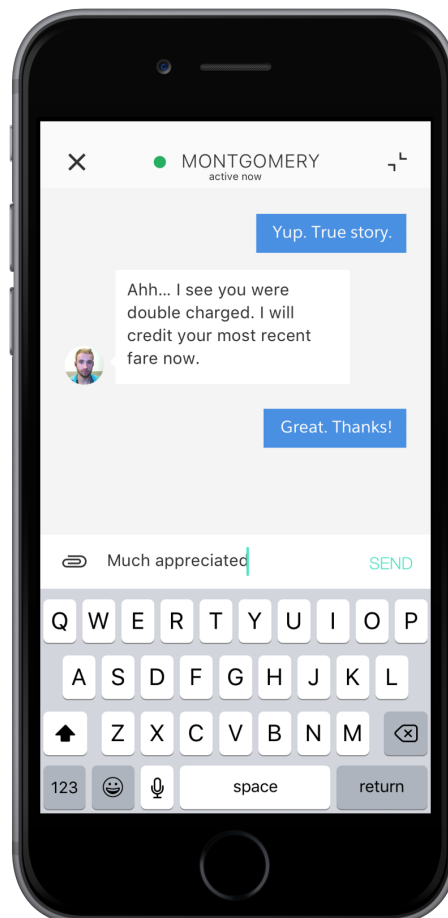
The unique button ID for your chat configuration. You can get the button ID by creating a Live Agent chat button (see [Create Chat Buttons](#) in the Live Agent help documentation), and instead of using the supplied JavaScript, just copy the `id` attribute. To get this value after creating a button, from **Setup**, search for **Chat Buttons** and select **Chat Buttons & Invitations**. Copy the `id` for the button from the JavaScript snippet.



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



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



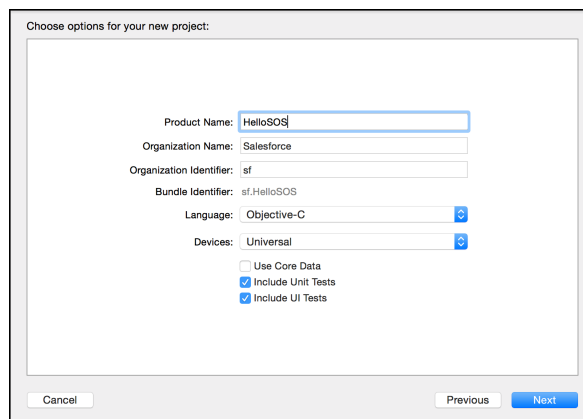
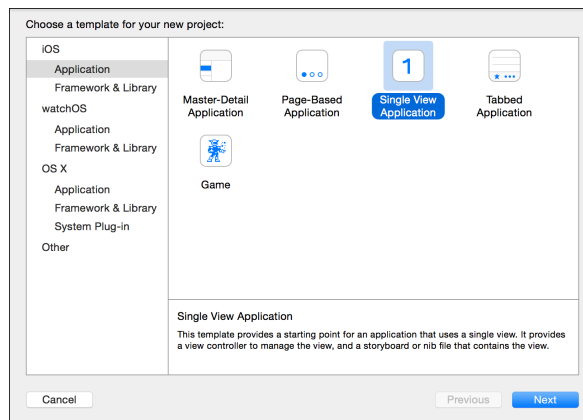
Get Started with SOS

See for yourself how easy and effective live video chat and screen sharing can be.

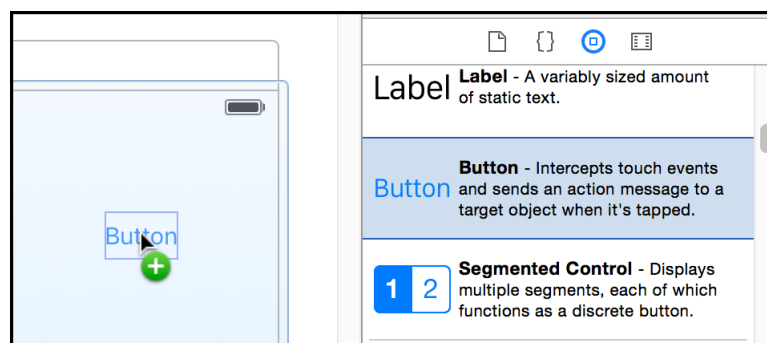
Before doing this tutorial, be sure that you've set up Service Cloud for SOS. See [Console Setup for SOS](#) for more information.

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

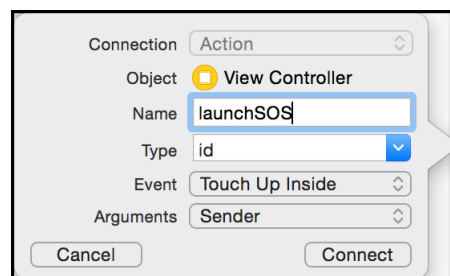
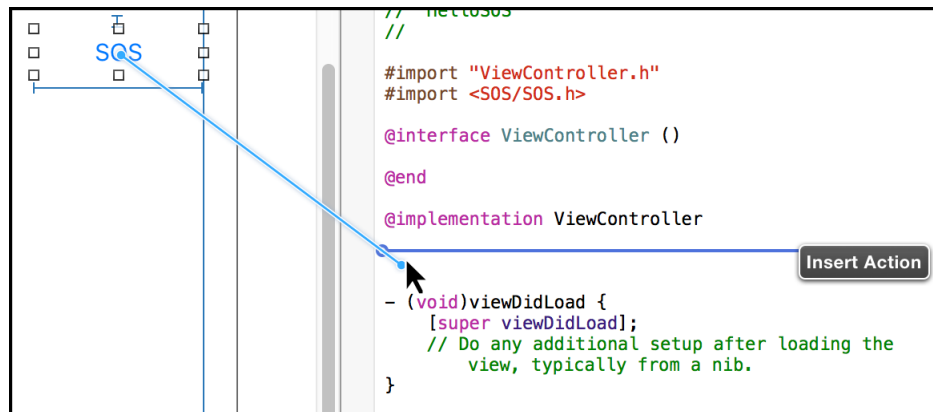
1. Create an Xcode project. For this example, let's make a Single View Application. Name it HelloSOS.



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



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



5. Import the SDK. Wherever you intend to use the SOS SDK, be sure to import the Service Common framework and the SOS framework.

In Objective-C:

```
@import ServiceCore;
#import ServicesSOS;
```

In Swift:

```
import ServiceCore
import ServicesSOS
```

6. Launch an SOS session from within the launchSOS method.

From the button action implementation, launch SOS using the `startSessionWithOptions` method on the `SOSSessionManager` shared instance.

In Objective-C:

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

    SOSOptions *options = [SOSOptions optionsWithLiveAgentPod:@"YOUR-POD-NAME"
                                                         orgId:@"YOUR-ORG-ID"
                                                         deploymentId:@"YOUR-DEPLOYMENT-ID"];

    [[SCServiceCloud sharedInstance].sos startSessionWithOptions:options];
}
```


In Swift:

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

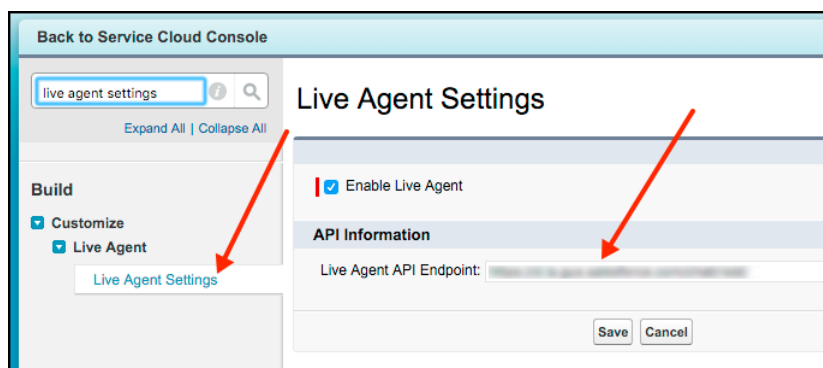
    let options = SOSOptions(liveAgentPod: "YOUR-POD-NAME",
                             orgId: "YOUR-ORG-ID",
                             deploymentId: "YOUR-DEPLOYMENT-ID")

    SCServiceCloud.sharedInstance().sos.startSessionWithOptions(options)
}
```

Fill in the placeholder text for the Live Agent pod, the org ID, and the deployment ID.

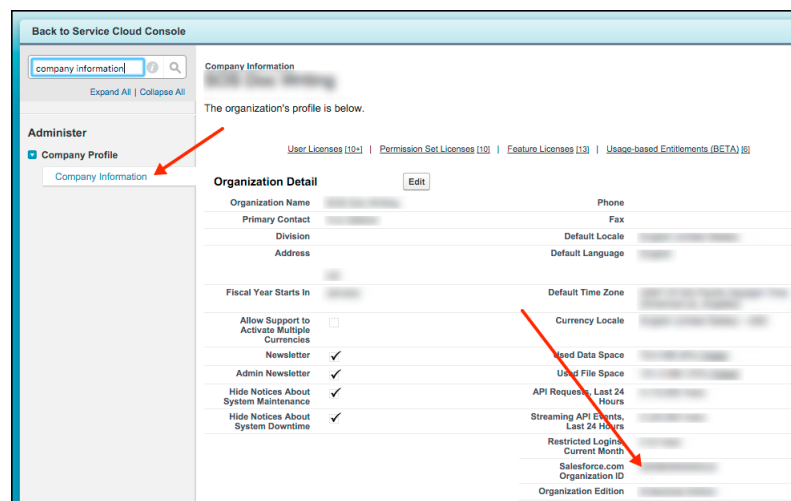
pod

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



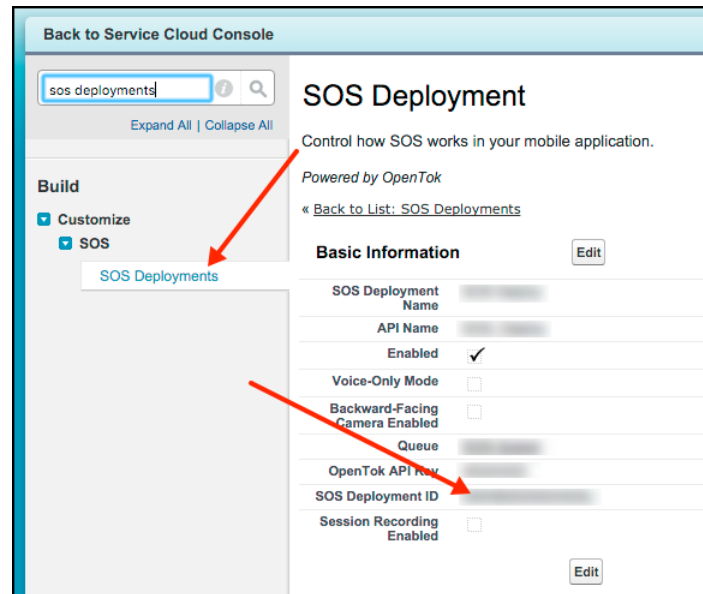
orgId

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

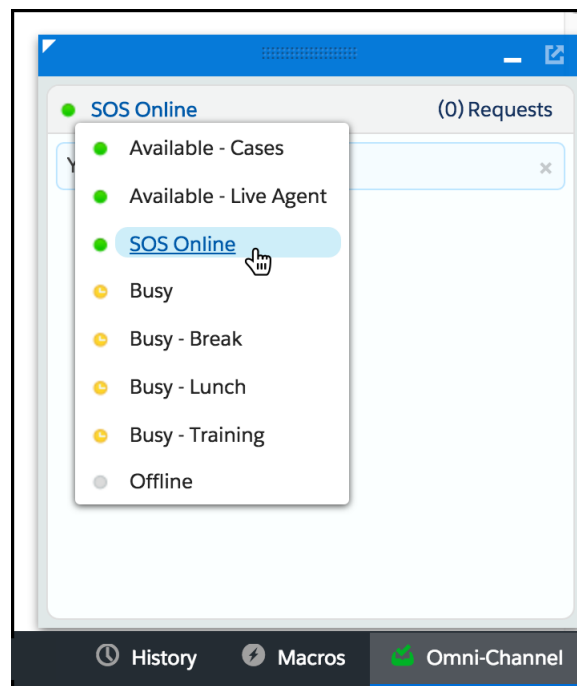


deploymentId

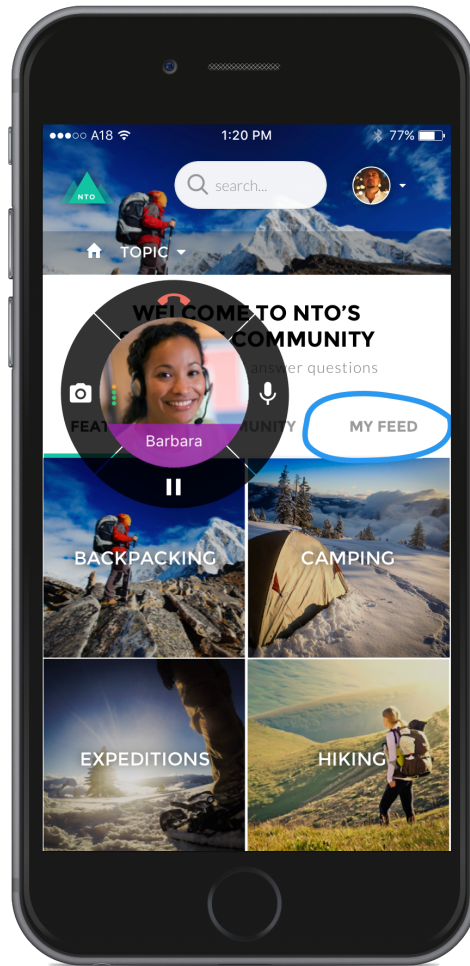
The unique ID of your SOS deployment. To get this value, from **Setup**, search for **SOS Deployments**, click the correct deployment and copy the **Deployment ID**.



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



Now you can build and run the app. When you tap the **SOS** button, the app requests an SOS session, which an agent can accept from the **Service Cloud Console**. From the console, you can chat with the customer, annotate things on their screen, and perform a two-way video session (if enabled).



Using Knowledge

Adding the Knowledge experience to your app.

[Knowledge Overview](#)

Learn about the Knowledge experience using the SDK.

[Quick Setup: Knowledge](#)

It's easy to get started with Knowledge for your iOS app. To set up Knowledge, enable the feature, point the shared instance to your community, and customize the look and feel.

[Knowledge as an Authenticated User](#)

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

[Customize the Presentation for Knowledge](#)

The simplest way to show and hide the Knowledge interface is by calling the `setInterfaceVisible` method on the `SCServiceCloud.knowledge` property. Alternatively, you can present the interface using a custom presentation. You can even manually display a Knowledge article yourself.

[Article Fetching and Caching](#)

By default, the SDK fetches knowledge articles as they are needed. These articles are then cached locally for faster access. However, using methods in `SCSKnowledgeManager`, you can pre-fetch articles to support offline access and other use cases.

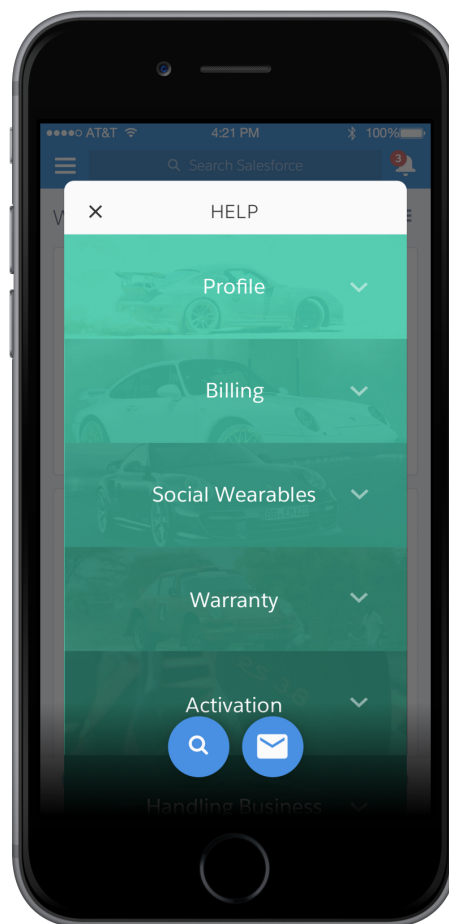
[Enable Case Management from Knowledge Interface](#)

By default, Case Management is disabled when a user accesses your Knowledge interface. However, you can enable Case Management so that a user can create a case with the tap of an action button within the Knowledge interface.

Knowledge Overview

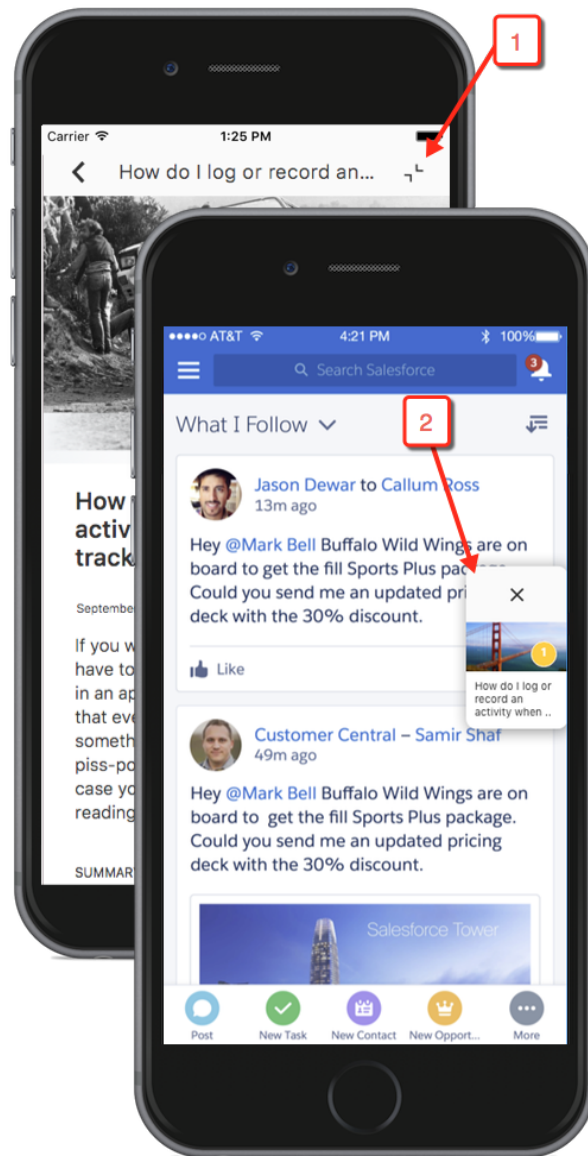
Learn about the Knowledge experience using the SDK.

The Knowledge feature in the SDK gives you access to your org's knowledge base directly from within your app. Once you [point your app](#) to your community URL with the right category group and root data category, you can [display your knowledge base](#) to your users.



By default, knowledge appears as a floating dialog on top of your app's existing content, though you can [customize the presentation](#) if you'd like. From the knowledge home, a user can navigate through articles that are organized by category. Articles are also searchable from within the app. When you [turn on the Case Management functionality](#), a user can create or manage cases using an action button from within the Knowledge interface.

When a user views an article, they can minimize it using the minimize button at the top right of the article (1) so that they can continue to navigate your app. The user can drag this thumbnail (2) to any part of the screen to improve visibility of the currently showing view. Tapping the X closes the article. Tap on any other part of the thumbnail to make it full screen again.



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.

Let's get started.

Quick Setup: Knowledge

It's easy to get started with Knowledge for your iOS app. To set up Knowledge, enable the feature, point the shared instance to your community, and customize the look and feel.

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

- Set up Service Cloud to work with Knowledge. To learn more, see [Cloud Setup for Knowledge](#).
- Installed the SDK. To learn more, see [Install the SDK](#).

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

1. Import the SDK. Wherever you intend to use the Knowledge SDK, be sure to import the Service Common framework and the Knowledge framework.

In Objective-C:

```
@import ServiceCore;
@import ServiceKnowledge;
```

In Swift:

```
import ServiceCore
import ServiceKnowledge
```

2. Point the SDK to your org from the `applicationDidFinishLaunchingWithOptions` method in your `UIApplicationDelegate` class.

To connect your app to your organization, create an [SCSServiceConfiguration](#) object containing the community URL, the data category group, and the root data category. Pass this object to the [SCServiceCloud](#) shared instance.

In Objective-C:

```
// Create configuration object with init params
SCSServiceConfiguration *config = [[SCSServiceConfiguration alloc]
initWithCommunity:[NSURL URLWithString:@"https://mycommunity.example.com"]
dataCategoryGroup:@"Regions"
rootDataCategory:@"All"];

// Perform any additional configuration here

// Pass configuration to shared instance
[SCServiceCloud sharedInstance].serviceConfiguration = config;
```

In Swift:

```
// Create configuration object with init params
let config = SCSServiceConfiguration(
    community: URL(string: "https://mycommunity.example.com")!,
    dataCategoryGroup: "Regions",
    rootDataCategory: "All")

// Perform any additional configuration here

// Pass configuration to shared instance
SCServiceCloud.sharedInstance().serviceConfiguration = config
```



Note: You can get the required parameters for this method from your Salesforce org. If your Salesforce Admin hasn't already set up Knowledge in Service Cloud or you need more guidance, see [Cloud Setup for Knowledge](#).

3. (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](#). Here's an example of how you can change the colors.

In Objective-C:

```
// Create appearance configuration object with some appearance changes
SCAppearanceConfiguration *appearance = [SCAppearanceConfiguration new];
[appearance setColor:[UIColor greenColor]
               forName:SCSAppearancePrimaryBrandColor];
[appearance setColor:[UIColor redColor]
               forName:SCSAppearanceBrandContrastColor];
[appearance setColor:[UIColor blackColor]
               forName:SCSAppearanceContrastPrimaryColor];

// Pass appearance configuration to shared instance
[SCServiceCloud sharedInstance].appearanceConfiguration = appearance;
```

In Swift:

```
// Create appearance configuration object with some appearance changes
let appearance = SCAppearanceConfiguration()
appearance.setColor(UIColor.green, forName: SCSAppearancePrimaryBrandColor)
appearance.setColor(UIColor.red, forName: SCSAppearanceBrandContrastColor)
appearance.setColor(UIColor.black, forName: SCSAppearanceContrastPrimaryColor)

// Pass appearance configuration to shared instance
SCServiceCloud.sharedInstance().appearanceConfiguration = appearance
```

You can also configure the strings used throughout the interface and which action buttons are visible when viewing your knowledge base.

See [SDK Customizations](#) for more info.

4. Enable the Knowledge feature from the `applicationDidFinishLaunchingWithOptions` method.

Use the `knowledge.enabled` property to enable Knowledge.

In Objective-C:

```
[SCServiceCloud sharedInstance].knowledge.enabled = YES;
```

In Swift:

```
SCServiceCloud.sharedInstance().knowledge.isEnabled = true
```

5. (Optional) Implement any of the Service SDK delegates.

SCServiceCloudDelegate

Access to general Service SDK events (for example, `willDisplayViewController`, `didDisplayViewController`, `shouldShowActionWithName`).

SCKnowledgeInterfaceDelegate

Access to Knowledge interface events (for example, `imageForArticle`, `imageForDataCategory`). See [Customize Images](#) for an example of using this delegate.

SCAppearanceConfigurationDelegate

Access to appearance-related events (for example, `appearanceConfigurationWillApplyUpdates`, `appearanceConfigurationDidApplyUpdates`).

6. Show the interface from your view controller.

You can show the interface as soon as the view controller loads, or start it from a UI action.

In Objective-C:

```
[[SCServiceCloud sharedInstance].knowledge setInterfaceVisible:YES
                                       animated:YES
                                       completion:nil];
```

In Swift:

```
SCServiceCloud.sharedInstance().knowledge.setInterfaceVisible(true,
                                                             animated: true,
                                                             completion: nil)
```

By default, the interface appears as a floating dialog. Alternatively, you can present the interface using a custom presentation. See [Customize the Presentation for Knowledge](#) for more info.

For instructions on launching the interface from a web view, see [Launch SDK from a Web View](#).

If you run into issues accessing your community, check out [Unable to Access My Community](#).



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

Set up the Knowledge interface within the `AppDelegate` implementation.

In Objective-C:

```
#import "AppDelegate.h"
#import ServiceCore;
#import ServiceKnowledge;

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // Create configuration object with init params
    SCSServiceConfiguration *config = [[SCSServiceConfiguration alloc]
        initWithCommunity:[NSURL URLWithString:@"https://mycommunity.example.com"]
        dataCategoryGroup:@"Regions"
        rootDataCategory:@"All"];

    // Pass configuration to shared instance
    [SCServiceCloud sharedInstance].serviceConfiguration = config;

    // Customize the appearance with the configuration object
    SCAppearanceConfiguration *appearanceConfiguration =
        [SCAppearanceConfiguration new];
    [appearanceConfiguration setColor:[UIColor greenColor]
        forName:SCSAppearancePrimaryBrandColor];
    [appearanceConfiguration setColor:[UIColor redColor]
        forName:SCSAppearanceBrandContrastColor];
    [appearanceConfiguration setColor:[UIColor blackColor]
        forName:SCSAppearanceContrastPrimaryColor];
    [SCServiceCloud sharedInstance].appearanceConfiguration =
        appearanceConfiguration;

    // Enable Knowledge
    [SCServiceCloud sharedInstance].knowledge.enabled = YES;
```



```

    return YES;
}
@end

```

In Swift:

```

import UIKit
import ServiceCore
import ServiceKnowledge

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication,
        didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?)
        -> Bool {

        // Create configuration object with init params
        let config = SCSServiceConfiguration(
            community: URL(string: "https://mycommunity.example.com")!,
            dataCategoryGroup: "Regions",
            rootDataCategory: "All")

        // Pass configuration to shared instance
        SCServiceCloud.sharedInstance().serviceConfiguration = config

        // Create appearance configuration object with some appearance changes
        let appearance = SCAppearanceConfiguration()
        appearance.setColor(UIColor.green, forName: SCAppearancePrimaryBrandColor)
        appearance.setColor(UIColor.red, forName: SCAppearanceBrandContrastColor)
        appearance.setColor(UIColor.black, forName: SCAppearanceContrastPrimaryColor)

        // Pass appearance configuration to shared instance
        SCServiceCloud.sharedInstance().appearanceConfiguration = appearance

        // Enable Knowledge
        SCServiceCloud.sharedInstance().knowledge.isEnabled = true

        return true
    }
}

```

Using the storyboard, add a button to the view. Then add a `Touch Up Inside` action in your `UIViewController` implementation with the name `showHelp`. When the button is clicked, make the Knowledge interface visible.

In Objective-C:

```

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

```

```

@implementation ViewController

- (IBAction)showHelp:(id)sender {

    // Show the Knowledge interface
    [[SCServiceCloud sharedInstance].knowledge setInterfaceVisible:YES
                                             animated:YES
                                             completion:nil];
}
@end

```

In Swift:

```

import UIKit
import ServiceCore
import ServiceKnowledge

class ViewController: UIViewController {

    @IBAction func showHelp(_ sender: AnyObject) {

        SCServiceCloud.sharedInstance().knowledge.setInterfaceVisible(true,
            animated: true,
            completion: nil)
    }
}

```

Knowledge as an Authenticated User

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

These instructions set up your knowledge base as an authenticated user. When you activate the Knowledge interface for authenticated users, they see knowledge content assigned to their user profile. If you do not want to authenticate users and prefer to let them see knowledge content accessible to guest users, see [Quick Setup: Knowledge](#) for instructions on accessing a public knowledge base.



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

1. Review the steps in [Quick Setup: Knowledge](#).

The basic steps for setting up and displaying the interface still apply for authenticated users.

2. Write the code to authenticate a user with your org.

You can authenticate in several ways.

- **Create normal user accounts on your Salesforce org.** If you have (or want to create) Salesforce user accounts for each user of your app, see [Digging Deeper into OAuth 2.0 on Force.com](#), which describes how to set up a connected app on Salesforce. For developers already using the Salesforce Mobile SDK, the [Mobile SDK Developer's Guide](#) contains instructions for [authenticating](#) with that SDK.
- **Use your own user credentials to create community portal users on Salesforce.** You can authenticate using existing credentials on your server. One approach is to create an API endpoint on your server that is accessible to your app. For each user,

your server can formulate unique information representing that user. Use this information in a JSON Web Token (JWT) payload. Sign and submit this information to the OAuth endpoint on your Salesforce org. Using Apex code configured within Salesforce, you can use the JWT information to identify a previously created community user or implicitly create a new user. That user's OAuth keys can be created and supplied back to your server.

- Documentation about using JWT for Salesforce authorization: [OAuth 2.0 JWT Bearer Token Flow](#).
- Documentation about creating a community portal user with an Apex trigger: [Apex Developer's Guide: Force.com Sites Examples](#).
- More documentation about creating a community portal user: [Authenticating Users on Force.com Sites](#).

The Service SDK does not contain any additional method to authenticate.

3. When authenticated, hold onto an `SFUserAccount` object with your credentials.

Whichever method you use to authenticate, you need an `SFUserAccount` object to pass to the Service SDK. This class is part of the `SalesforceKit` framework from the [Salesforce Mobile SDK](#). The account object must have correct information in its `credentials` property. If you're using the Mobile SDK to authenticate, the `SFOAuthCoordinator` instance you're given in your delegate handler contains a `credentials` property that you can use.

4. Implement `SCServiceCloudDelegate` and pass account information from within the `shouldAuthenticateService` method. The SDK calls this method whenever it plans to display a view that can support authenticated users.

The `SCServiceCloud` shared instance has a delegate you can implement.

In Objective-C:

```
[SCServiceCloud sharedInstance].delegate = self;
```

In Swift:

```
SCServiceCloud.sharedInstance().delegate = self
```

If you want an authenticated connection, implement the `shouldAuthenticateService` method from this delegate and return `YES/true`. This method is a good place to pass in the user account with the supplied completion block, which runs asynchronously.

In Objective-C:

```
- (BOOL)serviceCloud:(SCServiceCloud *)serviceCloud
    shouldAuthenticateService:(NSString *)service
        completion:(void (^)(SFUserAccount * _Nullable))completion {

    SFUserAccount* user = // TO DO: Get user account

    // After acquiring user information, call
    // this completion block to set the new user:
    completion(user);

    return YES;
}
```

In Swift:

```
func serviceCloud(serviceCloud: SCServiceCloud,
    shouldAuthenticateService service: String,
    completion: (SFUserAccount?) -> Void) -> Bool {
```

```

let user: SFUserAccount = // TO DO: Get user account

// After acquiring user information, call
// this completion block to set the new user:
completion(user)

return true
}

```

- For error handling, implement `serviceAuthenticationFailedWithError` from `SCServiceCloudDelegate`.

In Objective-C:

```

- (void)serviceCloud:(SCServiceCloud*)serviceCloud
  serviceAuthenticationFailedWithError:(NSError*)error
    forService:(NSString*)service {

    // TO DO: Inspect error and handle appropriately.
}

```

In Swift:

```

func serviceCloud(serviceCloud: SCServiceCloud,
  serviceAuthenticationFailedWithError error: Error,
    forService service: String) {

    // TO DO: Inspect error and handle appropriately.
}

```

The `service` param is `SCServiceTypeKnowledge` for Knowledge and `SCServiceTypeCases` for Case Management.

The `NSError` object contains information about the error. The `code` property on this object contains the error code. The following errors are the most common error codes you can encounter:

Error Code	Description
<code>SCServiceUserSessionExpiredOrInvalidError (401)</code>	Occurs when the session has expired or the ID is invalid. Use the refresh token to acquire another access token and then update the <code>account</code> property, as shown in the next step.
<code>SCServiceUserRequestRefusedError (403)</code>	Occurs when the user does not have sufficient access for the request. Verify that the logged in user has sufficient credentials.
<code>SCServiceUserResourceNotFoundError (404)</code>	Occurs when the requested resource was not found. Check the URI and other parameters for errors.

See [Status Codes and Error Responses](#) for a full set of possible error codes.

- When account information changes, update the `account` property.

Account information can change if the access token expires, the user logs out, another user logs in, and for various other reasons. You can update the [account property](#) on the `SCServiceCloud` shared instance. The Service SDK updates the interface to correspond with the updated account information. Preferably, instead of setting the property, use the [setAccount method](#), which allows you to specify a completion block for error handling.

In Objective-C:

```
SFUserAccount *user = // Get user account
[[SCServiceCloud sharedInstance]
    setAccount:user completion:^(NSError *error) {

    // TO DO: Handle error
}];
```

In Swift:

```
let user = // Get user account

SCServiceCloud.sharedInstance()
    .setAccount(user, completion: { (error: Error?) in

    // TO DO: Handle error
}))
```

After you authenticate users, they see only Knowledge content that is accessible to their user profile.

Customize the Presentation for Knowledge

The simplest way to show and hide the Knowledge interface is by calling the `setInterfaceVisible` method on the `SCServiceCloud.knowledge` property. Alternatively, you can present the interface using a custom presentation. You can even manually display a Knowledge article yourself.

Activating Interface Using the Default Presentation

Use the `setInterfaceVisible:animated:completion: method` to show the Knowledge interface using the default presentation. This method shows the interface as a floating dialog on top of your app's existing content. When the user drills into a detail screen, the interface automatically transitions to a full screen mode.

In Objective-C:

```
[[SCServiceCloud sharedInstance].knowledge setInterfaceVisible:YES
                                           animated:YES
                                           completion:nil];
```

In Swift:

```
SCServiceCloud.sharedInstance().knowledge.setInterfaceVisible(true,
    animated: true,
    completion: nil)
```

Activating Interface Using a Custom Transitioning Delegate

You can also present the interface using a custom transitioning animation and custom presentation. Just implement a `UIViewControllerTransitioningDelegate`.

1. Supply the `SCServiceCloud` shared instance with your `SCServiceCloudDelegate` implementation.

In Objective-C:

```
[SCServiceCloud sharedInstance].delegate = mySCServiceCloudDelegate;
```

In Swift:

```
SCServiceCloud.sharedInstance().delegate = mySCServiceCloudDelegate
```

2. Implement the `serviceCloud:transitioningDelegateForViewController:method` in your delegate and return a custom `UIViewControllerTransitioningDelegate` from this method.

In Objective-C:

```
- (NSObject<UIViewControllerTransitioningDelegate> *)
    serviceCloud:(SCServiceCloud *)serviceCloud
    transitioningDelegateForViewController:(UIViewController *)controller {

    // TO DO: Put your logic here and then return your transitioning delegate...

    return myTransitioningDelegate;
}
```

In Swift:

```
func serviceCloud(_ serviceCloud: SCServiceCloud,
    transitioningDelegateFor controller: UIViewController)
    -> UIViewControllerTransitioningDelegate? {

    // TO DO: Put your logic here and then return your transitioning delegate...

    return myTransitioningDelegate
}
```

Showing Article Content Manually

You can manually display an article by instantiating a `SCSArticleViewController`, specifying the article using the `article` property, and then presenting the view yourself. This method is useful if you want to display specific articles within your app. To learn more about this technique, see [Article Fetching and Caching](#).

Article Fetching and Caching

By default, the SDK fetches knowledge articles as they are needed. These articles are then cached locally for faster access. However, using methods in `SCSKnowledgeManager`, you can pre-fetch articles to support offline access and other use cases.

Article caching is valuable for several common use cases:

- Faster access to the most commonly viewed articles.
- Custom presentation of specific content.
- Offline access to some or all of your knowledge base.

There are several different ways to fetch knowledge articles. You can fetch all articles in a category. You can fetch articles based on a query. You can sort or limit the search results. And of course, you can also fetch specific articles by article ID.

There are several classes associated with article caching:

`SCSKnowledgeManager`

Manager class for interacting with Knowledge at the data level, including article caching functionality.

`SCSCategoryGroup`

Class that represents a category group.

SCSCategory

Class that represents a category or subcategory.

SCSMutableArticleQuery

Class used to build your article query. The immutable parent class is [SCSArticleQuery](#).

SCSArticle

Class that contains information about an article.

When using the caching feature, there are several areas to consider:

1. Fetching Categories
2. Querying for Articles
3. Downloading Content
4. Displaying Content

We've included an example after discussing these topics.

Fetching Categories

Since data categories are at the heart of knowledge articles, the categories for your org need to be downloaded before any interactions with Knowledge can be made. To see if categories have been cached, check the value for [hasFetchedCategories](#). If it returns NO, call [fetchAllCategoriesWithCompletionHandler:](#). When downloading individual articles, you don't need to make this call, but you won't be able to access content that requires information about data categories without performing this fetch.

In Objective-C:

```
SCSKnowledgeManager *knowledgeManager = [SCSKnowledgeManager defaultManager];
if (!knowledgeManager.hasFetchedCategories) {

    [knowledgeManager fetchAllCategoriesWithCompletionHandler:^(
        NSArray<SCSCategoryGroup * > * _Nullable categoryGroups,
        NSError * _Nullable error) {

        // TO DO: Get articles from each category using queryArticlesInCategory

    }];
}
```

In Swift:

```
let knowledgeManager = SCSKnowledgeManager.default()
if (!knowledgeManager.hasFetchedCategories()) {

    knowledgeManager.fetchAllCategories(completionHandler: {
        (categoryGroups: [SCSCategoryGroup]?, error: Error?) in

        // TO DO: Get articles from each category using queryArticlesInCategory

    })
}
```



Note: If your interface supports authenticated users, keep in mind that the user account information does not change for an preexisting instance of `SCSKnowledgeManager`. After you authenticate a user, call the static `defaultManager` method on `SCSKnowledgeManager` to get a new instance containing new user account information.

Querying for Articles (from Server, or Locally)

To fetch articles from the server for a given query, call `fetchArticlesWithQuery:`. To fetch local (already cached) articles for a given query, call `articlesMatchingQuery:`. These methods take an `SCSMutableArticleQuery` instance, which is an essential part of your fetch request. If you already fetched categories, you can drive this query based on a category; you can also directly query for an article based on its ID, or using another type of query.

When you create an `SCSMutableArticleQuery` instance, you can determine the type of query using the following properties:

Table 1: Query Types

Query Type	Property Name / Type	Notes
Articles matching article ID	<code>articleId: NSString</code>	The 18-character article ID. This property cannot be used with <code>searchTerm</code> , <code>queryMethod</code> , <code>sortOrder</code> , or <code>sortByField</code> .
Articles matching search term	<code>searchTerm: NSString</code>	This property cannot be used with <code>articleId</code> , <code>sortOrder</code> , or <code>sortByField</code> .
Articles within a set of categories	<code>categories: array of SCSCategory objects</code>	You must use the <code>queryMethod</code> filter described in the Query Filters table if you use this query type.

You can sort the query results with the following properties:

Table 2: Query Sort Properties

Query Sort Property	Property Name / Type / Default	Description
Sort using a specific sort order	<code>sortOrder: SCArticleSortOrder = Descending</code>	Whether to sort by ascending order or descending order.
Sort using a specific field type	<code>sortByField: SCArticleSortByField = LastPublishedDate</code>	Which field you want to sort by. For example: title, published date, view score.

You can filter the query results with the following properties:

Table 3: Query Filters

Query Filter	Property Name / Type / Default	Description
Number of articles to fetch	<code>pageSize: NSUInteger = 20</code>	The number of articles to retrieve. The server does not provide more than 100 articles at a time.
Filter selector for category (used with the <code>categories</code> property)	<code>queryMethod: SCQueryMethod = Below</code>	Whether you want the query to operate on just the specified categories (<code>At</code>), the categories and all their parent categories (<code>Above</code>), the categories and all their subcategories (<code>Below</code>). Typically, you'll

Query Filter	Property Name / Type / Default	Description
		<p>want to use <code>Below</code> to capture the specified category and its children.</p> <p>Be sure to have something specified in the <code>categories</code> property when using this filter.</p>

Before performing a query, you can check whether the query is valid using the `valid` property. A query is invalid when conflicting property values are specified. The following situations cause an invalid query: when `searchTerm` and `articleId` are both populated, when `searchTerm` and `sortByField` are both populated, when `searchTerm` and `sortOrder` are both populated, when `articleId` and `categories` are both populated, when `articleId` and `sortByField` are both populated, when `articleId` and `sortOrder` are both populated, and when `articleId` and `queryMethod` are both populated.

This code shows an example that queries using a search term with a limit of five articles per page.

In Objective-C:

```
SCSMutableArticleQuery *query = [SCSMutableArticleQuery new];
query.searchTerm = @"login issues";
query.pageSize = 5;

[knowledgeManager fetchArticlesWithQuery:query completion:^(
    NSArray<SCSArticle *> * _Nonnull articles,
    NSError * _Nullable error) {

    // TO DO: Download articles using downloadContentWithOptions

}];
```

In Swift:

```
let query = SCSMutableArticleQuery()
query.searchTerm = "login issues"
query.pageSize = 5

knowledgeManager.fetchArticles(with: query, completion: {
    (articles: [SCSArticle], error: Error?) in

    // TO DO: Download articles using downloadContentWithOptions

}))
```

Downloading Content

Once you've fetched the articles, you'll need to download them before displaying. Use the `downloadContentWithOptions:` method in the `SCSArticle` class to perform this function. This method caches the HTML content. It also fetches the article images, if you specify it to do so with the `options` parameter.

In Objective-C:

```
SCSArticle *article = // once you have an article
```

```
[article downloadContentWithOptions:
 (SCSArticleDownloadOptionRefetchArticleContent|SCSArticleDownloadOptionImages)
 completion:^(NSError * _Nullable error) {

    // TO DO: Handle completion

}]
```

In Swift:

```
let article = // once you have an article

let options: Int = SCSArticleDownloadOption.refetchArticleContent.rawValue |
                  SCSArticleDownloadOption.images.rawValue
article.downloadContent(withOptions: SCSArticleDownloadOption(rawValue:options)!,
 completion: { (error: Error?) in

    // TO DO: Handle completion

})
```

At this point, the article is downloaded and available for offline access.

If you're not sure if an article has already been downloaded, use the [isArticleContentDownloaded:](#) and [isAssociatedContentDownloaded:](#) methods to check before downloading.

Displaying Content

You've got two ways to present knowledge content:

1. Use the default presentation. If you use the [setInterfaceVisible:animated:completion: method](#), the SDK automatically displays the content, and it uses cached content before trying to get content online.
2. You can manually display an article by instantiating a [SCSArticleViewController](#), specifying the article using the [article](#) property, and then presenting the view yourself. This method is useful if you want to display specific articles within your app. When using this method, you can use the [SCSArticleViewControllerDelegate](#) class to listen for events.

In Objective-C:

```
// Get Knowledge Manager instance
SCSKnowledgeManager *knowledgeManager = [SCSKnowledgeManager defaultManager];

// Create query for a specific article
SCSMutableArticleQuery *query = [SCSMutableArticleQuery new];
query.articleId = @"TO_DO:QUERY_ID";

// Fetch article
[knowledgeManager fetchArticlesWithQuery:query completion:^(
    NSArray<SCSArticle *> * _Nonnull articles,
    NSError * _Nullable error) {

    if (error != nil) {
        // TO DO: Handle error
    }
    else if ([articles count] == 0) {
        // TO DO: Handle no results
    }
}
```

```

    }
    else {
        SCSArticle *article = articles[0];

        // Download article
        [article downloadContentWithOptions:
         (SCSArticleDownloadOptionRefetchArticleContent|SCSArticleDownloadOptionImages)
         completion:^(NSError* error) {

            if (error != nil) {
                // TO DO: Handle error
            } else {

                // Display view with article
                SCSArticleViewController *articleVC = [[SCSArticleViewController alloc] init];
                articleVC.article = article;
                [self presentViewController:articleVC animated:YES completion:nil];
            }
        }];
    }
}
}
}];

```

In Swift:

```

// Get Knowledge Manager instance
let knowledgeManager = SCSKnowledgeManager.default()

// Create query for a specific article
let query = SCSMutableArticleQuery()
query.articleId = "TO_DO:QUERY_ID"

// Fetch article
knowledgeManager.fetchArticles(with: query, completion: {
    (articles: [SCSArticle], error: Error?) in

    if (error != nil) {
        // TO DO: Handle error
    }
    else if (articles.count == 0) {
        // TO DO: Handle no results
    }
    else {
        let article:SCSArticle = articles[0]

        // Download article
        let options: Int = SCSArticleDownloadOption.refetchArticleContent.rawValue |
            SCSArticleDownloadOption.images.rawValue
        article.downloadContent(withOptions: SCSArticleDownloadOption(rawValue:options)!,
            completion: { (error: Error?) in

            if (error != nil) {
                // TO DO: Handle error
            } else {

                // Display view with article
            }
        })
    }
}

```

```

        let articleVC = SCSArticleViewController()
        articleVC.article = article
        self.present(articleVC, animated:true, completion: nil)
    }
    })
}
})

```



Example: This code shows an example of how to download the top three articles in each category.

In Objective-C:

```

SCSKnowledgeManager *knowledgeManager = [SCSKnowledgeManager defaultManager];

// Create an article download block
dispatch_block_t articleDownloadBlock = ^{

    // Get category group
    SCSCategoryGroup *categoryGroup =
        [knowledgeManager categoryGroupWithName:@"MY-CATEGORY-GROUP"];

    // Get root category from category group
    SCSCategory *rootCategory =
        [categoryGroup categoryWithName:@"MY-CATEGORY"];

    if (rootCategory != nil) {

        // Iterate through all categories in root category
        for (SCSCategory *category in rootCategory.childCategories) {

            // Build a query...
            SCSCMutableArticleQuery *query = [SCSCMutableArticleQuery new];

            // ... for the current category
            query.categories = [NSArray arrayWithObjects: category, nil];

            // ... containing the top 3 articles
            query.pageSize = 3;

            // And then fetch articles with that query
            [knowledgeManager fetchArticlesWithQuery:query completion:^(
                NSArray<SCSArticle *> * _Nonnull articles,
                NSError * _Nullable error) {

                // TO DO: Check for error

                // For each article object fetched
                for (SCSArticle *article in articles) {

                    // Fetch the contents
                    [article downloadContentWithOptions:
                        (SCSArticleDownloadOptionRefetchArticleContent |
                         SCSArticleDownloadOptionImages)
                     completion:nil];
                }
            }];
        }
    }
};

```

```

        }];
    }
}
};

// If we haven't fetched the categories
if (!knowledgeManager.hasFetchedCategories) {

    // Then first fetch the categories
    [knowledgeManager fetchAllCategoriesWithCompletionHandler:^(
        (NSArray<SCSCategoryGroup *> * _Nullable categoryGroups,
        NSError * _Nullable error) {

        // TO DO: Check for error

        // And then download the articles
        articleDownloadBlock();

    }]);
} else {

    // Download the articles
    articleDownloadBlock();
}

```

In Swift:

```

let knowledgeManager = SCSSKnowledgeManager.default()

// Create an article download block
let articleDownloadBlock = {

    // Get category group
    let categoryGroup = knowledgeManager.categoryGroup(withName: "MY-CATEGORY-GROUP")

    // Get root category from category group
    let rootCategory = categoryGroup?.category(withName: "MY-CATEGORY")

    if (rootCategory != nil) {

        // Iterate through all categories in root category
        for category in (rootCategory!.childCategories) {

            // Build a query...
            let query = SCSSMutableArticleQuery()

            // ... for the current category
            query.categories = [category]

            // ... containing the top 3 articles
            query.pageSize = 3

            // And then fetch articles with that query
            knowledgeManager.fetchArticles(with: query, completion:
            { (articles: [SCSArticle], error: Error?) in

```

```

        // TO DO: Check for error

        // For each article object fetched
        for article in articles {

            // Fetch the contents
            let options: Int = SCSEArticleDownloadOption.refetchArticleContent.rawValue
|
            SCSEArticleDownloadOption.images.rawValue
            article.downloadContent(withOptions: SCSEArticleDownloadOption(rawValue:
options)!,
                                completion: nil)
        }
    })
}
}

// If we haven't fetched the categories
if (!knowledgeManager.hasFetchedCategories()) {

    // Then first fetch the categories
    knowledgeManager.fetchAllCategories(completionHandler:
    { (categoryGroups: [SCSCategoryGroup]?, error: Error?) in

        // TO DO: Check for error

        // And then download the articles
        articleDownloadBlock();
    })
}
else {
    // Download the articles
    articleDownloadBlock();
}
}

```

Enable Case Management from Knowledge Interface

By default, Case Management is disabled when a user accesses your Knowledge interface. However, you can enable Case Management so that a user can create a case with the tap of an action button within the Knowledge interface.

Before you can enable case management from within the knowledge interface, be sure you have correctly set up the knowledge interface. To learn more, see [Quick Setup: Knowledge](#).

1. Enable the Case Management feature.

Use the `cases.enabled` property to enable the Case Publisher feature.

In Objective-C:

```
[SCServiceCloud sharedInstance].cases.enabled = YES;
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.isEnabled = true
```

2. (Optional) Authenticate the user so that they can access their list of existing cases.

If you also want to allow a user to access their list of existing cases, you'll need to authenticate the user. To learn more, see [Case Management as an Authenticated User](#).

Using Case Management

Adding the Case Management experience to your app.

[Case Management Overview](#)

Learn about the Case Publisher and Case Management experience using the SDK.

[Quick Setup: Case Publisher as a Guest User](#)

It's easy for a user to create cases from your iOS app. To set up case publisher, enable the feature, point the shared instance to your community, and customize the look and feel.

[Case Management as an Authenticated User](#)

To manage existing cases, a user must authenticate with your org. Once authenticated, the user can both create and manage cases from your app.

[Customize the Presentation for Case Management](#)

The simplest way to show and hide the Case Management interface is by calling the `setInterfaceVisible` method on the `SCServiceCloud.cases` property. Alternatively, you can present the interface using a custom presentation. You can even manually control the Case Management view controllers yourself.

[Send Custom Data Using Hidden Fields](#)

You can hide specific Case-related fields in your Case Management views. This behavior is useful if you want to pass information to Service Cloud that does not require user input and that the user shouldn't see. To make this happen, implement the view controller delegates and specify the hidden fields.

[Configure Case Deflection](#)

When a user enters information about a new case – if you have a knowledge base available to that user – the SDK automatically searches that knowledge base for relevant articles and offers them to the user. You have the ability to turn this feature on and off, as well as control which case publisher fields are used to search content.

[Customize the Case Publisher Result View](#)

By default, when a user submits a new case from the Case Publisher screen, a standard success view appears. If you want to provide your users with more specific guidance after a case is created, one solution is to customize the view's message text and default image. If you'd like more control over what is displayed, you can present your own view by implementing the `viewForResult` method in the `SCSCasePublisherViewControllerDelegate` class.

[Push Notifications for Case Activity](#)

Using Salesforce's push notification implementation guide, it's easy to set up notifications in your app when there is activity associated with a case for an authenticated user.

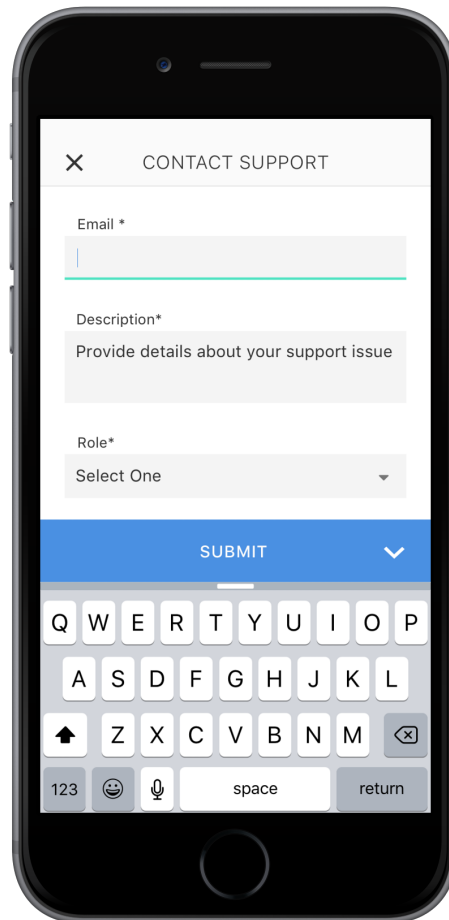
[Automated Email Responses](#)

Create automated email responses when a case is submitted from your app.

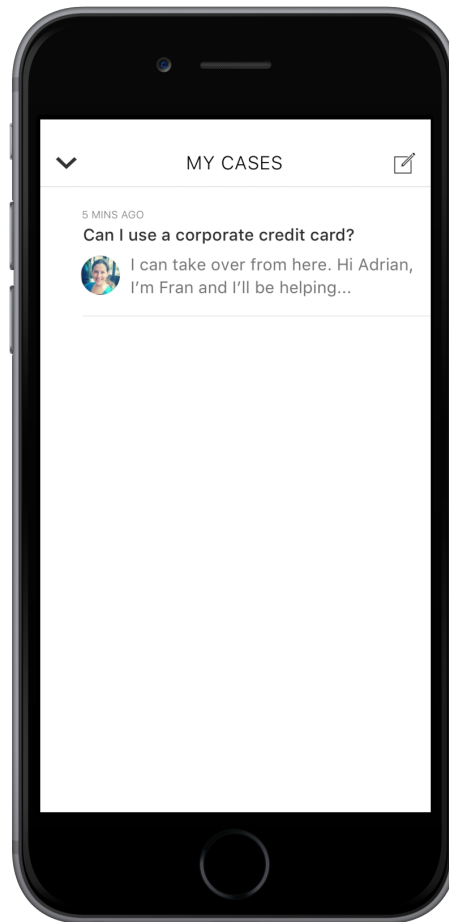
Case Management Overview

Learn about the Case Publisher and Case Management experience using the SDK.

The Case Management feature in the SDK allows your users to create and manage cases. Once you [point your app](#) to your community URL, you can [display the case management interface](#) to your users. If you don't authenticate the user, they can still [create new cases](#) using the guest user profile. A user creates a new case with the **Case Publisher** screen:



If you [authenticate the user](#), they can also view and manage their list of cases. In the [default 'guest user' flow](#), launching the interface causes the **Case Publisher** screen to appear. From this screen, a user can create a case as an anonymous guest user. In the default 'authenticated user' flow, launching the interface causes the **Case List** screen to appear. From this screen, a user can inspect an existing case (which launches the **Case Details** screen), or create a new case (which launches the **Case Publisher** screen).



If you'd prefer, you can [manually control](#) the Case Management view controllers.

For authenticated users, you can [set up notifications](#) so they are notified when there's a new post associated with one of their existing cases. You can even set it up so that the **Case Details** screen [automatically appears](#) with the latest case activity.

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.

Let's get started.

Quick Setup: Case Publisher as a Guest User

It's easy for a user to create cases from your iOS app. To set up case publisher, enable the feature, point the shared instance to your community, and customize the look and feel.

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

- Set up Service Cloud to work with Case Management. To learn more, see [Cloud Setup for Case Management](#).
- Installed the SDK. To learn more, see [Install the SDK](#).

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

These instructions allow you to set up case publisher as a guest user. This functionality allows a user to publish a new case. However, a guest user cannot manage existing cases. To manage cases, you'll need to authenticate the user, which requires a few more steps. To learn more about the authenticated user setup, see [Case Management as an Authenticated User](#).

1. Import the SDK. Wherever you intend to use the Case Management SDK, be sure to import the Service Common framework and the Case Management framework.

In Objective-C:

```
@import ServiceCore;
#import ServiceCases;
```

In Swift:

```
import ServiceCore
import ServiceCases
```

2. Point the SDK to your org from the `applicationDidFinishLaunchingWithOptions` method in your `UIApplicationDelegate` class.

To connect your application to your organization, create an `SCSServiceConfiguration` object containing the community URL and pass this object to the `SCServiceCloud` shared instance.

In Objective-C:

```
// Create configuration object with your community URL
SCSServiceConfiguration *config = [[SCSServiceConfiguration alloc]
initWithCommunity:[NSURL URLWithString:@"https://mycommunity.example.com"]];

// Pass configuration to shared instance
[SCServiceCloud sharedInstance].serviceConfiguration = config;
```

In Swift:

```
// Create configuration object with your community URL
let config = SCSServiceConfiguration(
    community: URL(string: "https://mycommunity.example.com")!)

// Pass configuration to shared instance
SCServiceCloud.sharedInstance().serviceConfiguration = config
```

You can get the community URL from your Salesforce org. From **Setup**, search for **All Communities**, and copy the URL for the desired community. For more help, see [Cloud Setup for Case Management](#).



Note: If you plan to access Knowledge in addition to Case Management, use the `initWithCommunity:dataCategoryGroup:rootDataCategory:method` instead. This method sets up data categories in addition to setting the community URL. See [Quick Setup: Knowledge](#) in the Knowledge section for more info.

3. Assign a global action to the Case Management interface. The global action determines the fields shown when a user creates a case. To configure the fields shown when creating a case, specify the global action name in the `caseCreateActionName` property. This code snippet illustrates how to associate the case publisher feature with the **New Case** global action layout, which is one of the default actions provided in most orgs.

In Objective-C:

```
[SCServiceCloud sharedInstance].cases.caseCreateActionName = @"NewCase";
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.caseCreateActionName = "NewCase"
```

You can get the global action name from your Salesforce org. From **Setup**, search for **Global Actions**, and copy the name of the desired quick action. For more help, see [Cloud Setup for Case Management](#).



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

4. Enable the Case Management feature from the `applicationDidFinishLaunchingWithOptions` method. Use the `cases.enabled` property to enable Case Management.

In Objective-C:

```
[SCServiceCloud sharedInstance].cases.enabled = YES;
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.isEnabled = true
```

5. (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](#) for more info.

6. (Optional) Implement any of the Service SDK delegates.

SCServiceCloudDelegate

Access to general Service SDK events (for example, `willDisplayViewController`, `didDisplayViewController`, `shouldShowActionWithName`).

SCAppearanceConfigurationDelegate

Access to appearance-related events (for example, `appearanceConfigurationWillApplyUpdates`, `appearanceConfigurationDidApplyUpdates`).

7. Show the interface from your view controller.

You can show the interface as soon as the view controller loads, or start it from a UI action.



Note: If you show the interface as a guest user, the case publisher screen appears. If you choose to authenticate first (see [Case Management as an Authenticated User](#)), then the case list screen is the first thing to appear.

In Objective-C:

```
[[SCServiceCloud sharedInstance].cases setInterfaceVisible:YES
                                     animated:YES
                                     completion:nil];
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.setInterfaceVisible(true,
                                                         animated: true,
                                                         completion: nil)
```

By default, the interface appears as a floating dialog. Alternatively, you can present the interface using a custom presentation. You can even manually control the Case Management view controllers yourself. See [Customize the Presentation for Case Management](#) for more info.

For instructions on launching the interface from a web view, see [Launch SDK from a Web View](#).



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

Set up the Case Management interface within the AppDelegate implementation.

In Objective-C:

```
#import "AppDelegate.h"
#import ServiceCore;
#import ServiceCases;

@implementation AppDelegate

- (BOOL)application:(UIApplication *)application
    didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // Point the SDK to your community
    SCSServiceConfiguration *config = [[SCSServiceConfiguration alloc]
        initWithCommunity:[NSURL URLWithString:@"https://mycommunity.example.com"]];
    [SCServiceCloud sharedInstance].serviceConfiguration = config;

    // Assign global action to use for case layout
    [SCServiceCloud sharedInstance].cases.caseCreateActionName = @"NewCase";

    // Enable Case Management
    [SCServiceCloud sharedInstance].cases.enabled = YES;

    return YES;
}

@end
```

In Swift:

```
import UIKit
import ServiceCore
import ServiceCases

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {

        // Point the SDK to your community
        let config = SCSServiceConfiguration(commUNITY: URL(string:
            "https://mycommunity.example.com")!)
        SCServiceCloud.sharedInstance().serviceConfiguration = config

        // Assign global action to use for case layout
        SCServiceCloud.sharedInstance().cases.caseCreateActionName = "NewCase"

        // Enable Case Management
        SCServiceCloud.sharedInstance().cases.isEnabled = true
    }
}
```

```

        return true
    }
}

```

Using the storyboard, add a button to the view. Then add a `Touch Up Inside` action in your `UIViewController` implementation with the name `showHelp`. When the button is clicked, make the Case Management interface visible.

In Objective-C:

```

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

@implementation ViewController

- (IBAction)showHelp:(id)sender {

    // Show the Case Management interface
    [[SCServiceCloud sharedInstance].cases setInterfaceVisible:YES
                                                animated:YES
                                                completion:nil];
}

@end

```

In Swift:

```

import UIKit
import ServiceCore
import ServiceKnowledge

class ViewController: UIViewController {

    @IBAction func showHelp(_ sender: AnyObject) {

        SCServiceCloud.sharedInstance().cases.setInterfaceVisible(true,
            animated: true,
            completion: nil)
    }
}

```

Case Management as an Authenticated User

To manage existing cases, a user must authenticate with your org. Once authenticated, the user can both create and manage cases from your app.

These instructions allow you to set up case management as an authenticated user. When you activate the Case Management interface for an authenticated user, a list of their existing cases appears initially. From there, they can inspect an existing case, or they can create a new case. If you do not want to authenticate users, and you prefer to let them create cases as a guest user, see [Quick Setup: Case Publisher as a Guest User](#).

1. Review the steps in [Quick Setup: Case Publisher as a Guest User](#).

The basic steps for setting up and displaying the interface still apply for authenticated users.

2. Write the code to authenticate a user with your org.

You can authenticate in several ways.

- **Create normal user accounts on your Salesforce org.** If you have (or want to create) Salesforce user accounts for each user of your app, see [Digging Deeper into OAuth 2.0 on Force.com](#), which describes how to set up a connected app on Salesforce. For developers already using the Salesforce Mobile SDK, the [Mobile SDK Developer's Guide](#) contains instructions for [authenticating](#) with that SDK.
- **Use your own user credentials to create community portal users on Salesforce.** You can authenticate using existing credentials on your server. One approach is to create an API endpoint on your server that is accessible to your app. For each user, your server can formulate unique information representing that user. Use this information in a JSON Web Token (JWT) payload. Sign and submit this information to the OAuth endpoint on your Salesforce org. Using Apex code configured within Salesforce, you can use the JWT information to identify a previously created community user or implicitly create a new user. That user's OAuth keys can be created and supplied back to your server.
 - Documentation about using JWT for Salesforce authorization: [OAuth 2.0 JWT Bearer Token Flow](#).
 - Documentation about creating a community portal user with an Apex trigger: [Apex Developer's Guide: Force.com Sites Examples](#).
 - More documentation about creating a community portal user: [Authenticating Users on Force.com Sites](#).

The Service SDK does not contain any additional method to authenticate.

3. When authenticated, hold onto an `SFUserAccount` object with your credentials.

Whichever method you use to authenticate, you need an `SFUserAccount` object to pass to the Service SDK. This class is part of the `SalesforceKit` framework from the [Salesforce Mobile SDK](#). The account object must have correct information in its `credentials` property. If you're using the Mobile SDK to authenticate, the `SFOAuthCoordinator` instance you're given in your delegate handler contains a `credentials` property that you can use.

4. Implement `SCServiceCloudDelegate` and pass account information from within the `shouldAuthenticateService` method.

The `SCServiceCloud` shared instance has a delegate you can implement.

In Objective-C:

```
[SCServiceCloud sharedInstance].delegate = self;
```

In Swift:

```
SCServiceCloud.sharedInstance().delegate = self
```

If you want an authenticated connection, implement the `shouldAuthenticateService` method from this delegate and return `YES/true`. This method is a good place to pass in the user account with the supplied completion block, which runs asynchronously.

In Objective-C:

```
- (BOOL)serviceCloud:(SCServiceCloud *)serviceCloud
    shouldAuthenticateService:(NSString *)service
        completion:(void (^)(SFUserAccount * _Nullable))completion {

    SFUserAccount* user = // TO DO: Get user account

    // After acquiring user information, call
    // this completion block to set the new user:
```

```
completion(user);

return YES;
}
```

In Swift:

```
func serviceCloud(serviceCloud: SCServiceCloud,
                  shouldAuthenticateService service: String,
                  completion: (SFUserAccount?) -> Void) -> Bool {

    let user: SFUserAccount = // TO DO: Get user account

    // After acquiring user information, call
    // this completion block to set the new user:
    completion(user)

    return true
}
```

- For error handling, implement `serviceAuthenticationFailedWithError` from `SCServiceCloudDelegate`.

In Objective-C:

```
- (void)serviceCloud:(SCServiceCloud*)serviceCloud
    serviceAuthenticationFailedWithError:(NSError*)error
    forService:(NSString*)service {

    // TO DO: Inspect error and handle appropriately.
}
```

In Swift:

```
func serviceCloud(serviceCloud: SCServiceCloud,
                  serviceAuthenticationFailedWithError error: Error,
                  forService service: String) {

    // TO DO: Inspect error and handle appropriately.
}
```

The `service` param is `SCServiceTypeKnowledge` for Knowledge and `SCServiceTypeCases` for Case Management. The `NSError` object contains information about the error. The `code` property on this object contains the error code. The following errors are the most common error codes you can encounter:

Error Code	Description
<code>SCServiceUserSessionExpiredOrInvalidError (401)</code>	Occurs when the session has expired or the ID is invalid. Use the refresh token to acquire another access token and then update the <code>account</code> property, as shown in the next step.
<code>SCServiceUserRequestRefusedError (403)</code>	Occurs when the user does not have sufficient access for the request. Verify that the logged in user has sufficient credentials.
<code>SCServiceUserResourceNotFoundError (404)</code>	Occurs when the requested resource was not found. Check the URI and other parameters for errors.

See [Status Codes and Error Responses](#) for a full set of possible error codes.

6. Assign a Cases view name to allow authenticated users to see a list of all their existing cases.

To show a list of cases, specify the unique name for your preferred Cases view from your Salesforce org. To get this value, access the **Cases** tab in your org, pick the desired **View**, select **Go!** to see that view, and then select **Edit** to edit the view. From the edit window, you can see the **View Unique Name**. Use this value when you specify the `caseListName` in the SDK. For more help, see [Cloud Setup for Case Management](#).

In Objective-C:

```
[SCServiceCloud sharedInstance].cases.caseListName = @"AllOpenCases";
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.caseListName = "AllOpenCases"
```

7. (Optional) Incorporate notifications whenever there is a new text post on an existing case.

Using Salesforce's push notification implementation guide, it's easy to set up notifications in your app when there is activity associated with a case for an authenticated user. To learn more, see [Push Notifications for Case Activity](#).

8. When account information changes, update the `account` property.

Account information can change if the access token expires, the user logs out, another user logs in, and for various other reasons. You can update the [account property](#) on the `SCServiceCloud` shared instance. The Service SDK updates the interface to correspond with the updated account information. Preferably, instead of setting the property, use the [setAccount method](#), which allows you to specify a completion block for error handling.

In Objective-C:

```
SFUserAccount *user = // Get user account
[[SCServiceCloud sharedInstance]
    setAccount:user completion:^(NSError *error) {

    // TO DO: Handle error
}];
```

In Swift:

```
let user = // Get user account

SCServiceCloud.sharedInstance()
    .setAccount(user, completion: { (error: Error?) in

    // TO DO: Handle error
}))
```

After you authenticate users, they have access to the full Case Management functionality, with easy access to information about their existing cases.

Customize the Presentation for Case Management

The simplest way to show and hide the Case Management interface is by calling the `setInterfaceVisible` method on the `SCServiceCloud.cases` property. Alternatively, you can present the interface using a custom presentation. You can even manually control the Case Management view controllers yourself.

In the [default 'guest user' flow](#), launching the interface causes the **Case Publisher** screen to appear. From this screen, a user can create a case as an anonymous guest user. In the default 'authenticated user' flow, launching the interface causes the **Case List** screen to appear. From this screen, a user can inspect an existing case (which launches the **Case Details** screen), or create a new case (which launches the **Case Publisher** screen). If you don't want to use a default user flow, you can manually show the Case Management view controllers.



Note: The fields in the Case Publisher screen are determined by the global action specified in your org. These fields are also shown at the top of the Case Details screen. See the global action step in [Quick Setup: Case Publisher as a Guest User](#) for more info.

Activating Interface Using the Default Presentation

Use the [setInterfaceVisible:animated:completion: method](#) to show the Case Management interface using the default presentation.

In Objective-C:

```
[SCServiceCloud sharedInstance].cases setInterfaceVisible:YES
                                     animated:YES
                                     completion:nil];
```

In Swift:

```
SCServiceCloud.sharedInstance().cases.setInterfaceVisible(true,
                                                         animated: true,
                                                         completion: nil)
```

If you just want to display the default Case Management experience, that's all you need to do. But if you'd like to access the associated view controllers, or their delegates, implement [SCServiceCloudDelegate](#) and supply that implementation to [SCServiceCloud](#).

In Objective-C:

```
[SCServiceCloud sharedInstance].delegate = self;
```

In Swift:

```
SCServiceCloud.sharedInstance().delegate = self
```

In your [SCServiceCloudDelegate](#) implementation, use the [willDisplayViewController](#) method to find out if it's a view controller you're interested in, and if so, assign the view controller delegate.

In Objective-C:

```
-(void)serviceCloud:(SCServiceCloud *)serviceCloud
    willDisplayViewController:(UIViewController *)controller
        animated:(BOOL)animated {

    // Case Publisher View
    if ([controller isKindOfClass:[SCSCasePublisherViewController class]]) {
        SCSCasePublisherViewController *casePublisherController =
            (SCSCasePublisherViewController *)controller;

        // TO DO: Implement SCSCasePublisherViewControllerDelegate
        casePublisherController.delegate = self;

    // Case Detail View
    } else if ([controller isKindOfClass:[SCSCaseDetailViewController class]]) {
        SCSCaseDetailViewController *caseDetailController =
```

```

        (SCSCaseDetailViewController *)controller;

        // TO DO: Implement SCSCaseDetailViewControllerDelegate
        caseDetailController.delegate = self;

        // Case List View
    } else if ([controller isKindOfClass:[SCSCaseListViewController class]]) {
        SCSCaseListViewController *caseListController =
            (SCSCaseDetailViewController *)controller;

        // TO DO: Implement SCSCaseListViewControllerDelegate
        caseListController.delegate = self;
    }
}

```

In Swift:

```

func serviceCloud(_ serviceCloud: SCServiceCloud,
                  willDisplay controller: UIViewController,
                  animated: Bool) {

    // Case Publisher View
    if (controller is SCSCasePublisherViewController) {
        let publisherController = controller as! SCSCasePublisherViewController

        // TO DO: Implement SCSCasePublisherViewControllerDelegate
        publisherController.delegate = self

    // Case Detail View
    } else if (controller is SCSCaseDetailViewController) {
        let detailController = controller as! SCSCaseDetailViewController

        // TO DO: Implement SCSCaseDetailViewControllerDelegate
        detailController.delegate = self

    // Case List View
    } else if (controller is SCSCaseListViewController) {
        let listController = controller as! SCSCaseListViewController

        // TO DO: Implement SCSCaseListViewControllerDelegate
        listController.delegate = self
    }
}

```

You now have access to the view controllers and their delegates.

Activating Interface Using a Custom Transitioning Delegate

You can also present the interface using a custom transitioning animation and custom presentation. Just implement a `UIViewControllerTransitioningDelegate`.

1. Supply the [SCServiceCloud shared instance](#) with your `SCServiceCloudDelegate` implementation.

In Objective-C:

```
[SCServiceCloud sharedInstance].delegate = mySCServiceCloudDelegate;
```

In Swift:

```
SCServiceCloud.sharedInstance().delegate = mySCServiceCloudDelegate
```

2. Implement the `serviceCloud:transitioningDelegateForViewController:method` in your delegate and return a custom `UIViewControllerTransitioningDelegate` from this method.

In Objective-C:

```
- (NSObject<UIViewControllerTransitioningDelegate> *)
    serviceCloud:(SCServiceCloud *)serviceCloud
    transitioningDelegateForViewController:(UIViewController *)controller {

    // TO DO: Put your logic here and then return your transitioning delegate...

    return myTransitioningDelegate;
}
```

In Swift:

```
func serviceCloud(_ serviceCloud: SCServiceCloud,
    transitioningDelegateFor controller: UIViewController)
    -> UIViewControllerTransitioningDelegate? {

    // TO DO: Put your logic here and then return your transitioning delegate...

    return myTransitioningDelegate
}
```

Showing the View Controllers Manually

Instead of having the SDK manage the case management flow, you can instantiate any of the view controllers and display it manually. When instantiating a view controller, be sure to implement the associated delegate and pass that delegate to the view controller (using the `delegate` property).

Feature	View Controller	Delegate
Case Publisher — for creating new cases	SCSCasePublisherViewController	SCSCasePublisherViewControllerDelegate
Case List — for viewing a list of a user's cases	SCSCaseListViewController	SCSCaseListViewControllerDelegate
Case Details — for inspecting the details of one case	SCSCaseDetailViewController	SCSCaseDetailViewControllerDelegate



Note: If you manually display the **Case List** view controller ([SCSCaseListViewController](#)), you'll also need to manually display the **Case Detail** view controller ([SCSCaseDetailViewController](#)). When a user selects a case from the case list, present your **Case Detail** view controller from the `caseList:selectedCaseWithId:` method in your [SCSCaseListViewControllerDelegate](#) implementation. If you don't do this, nothing will happen when a user selects a specific case in the case list!

For a use case that involves custom view controllers, see [Send Custom Data Using Hidden Fields](#).

Send Custom Data Using Hidden Fields

You can hide specific Case-related fields in your Case Management views. This behavior is useful if you want to pass information to Service Cloud that does not require user input and that the user shouldn't see. To make this happen, implement the view controller delegates and specify the hidden fields.

Before getting started, you'll need to be sure that your global action layout (that you specified with the [caseCreateActionName](#) property as described in [Quick Setup: Case Publisher as a Guest User](#)) contains the fields you want to hide. This layout is used for both the Case Publisher screen (where users can fill in values for the fields) and the Case Details screen (where users can view the field values). To learn more about quick actions, see [Create Global Quick Actions](#) in Salesforce Help.

Once you have the correct fields in your layout, there are two approaches to hiding specific fields:

1. Use the default Case Management view controllers, but implement the view controller delegates.
2. Instantiate (and display) the view controllers yourself, and also implement the view controller delegates.

The first method is useful if you don't want to manually display the views yourself—you simply want to hide specific fields. The second method is useful if you want more control over how and when to display the views. Details about each of the two methods are described later in this section.

Regardless of which method you choose, you'll need to implement the Case Management view controller delegates.

Implementing the Delegates

To affect which fields are shown in your Case Publisher view, implement the [SCSCasePublisherViewControllerDelegate](#). Use the [fieldsToHideFromCaseFields](#) to specify which fields to hide. This method passes you an array of available fields to hide. This array contains the unique API name for each field (which is not necessarily the label text for the field). From this array, return a set of fields to hide.

In Objective-C:

```
- (NSSet *)casePublisher:(SCSCasePublisherViewController *)publisher
    fieldsToHideFromCaseFields:(NSArray *)availableFields {

    NSSet *hideFieldsSet = [NSSet setWithObjects: @"MyHiddenField",
                                                @"MyOtherHiddenField", nil];

    return hideFieldsSet;
}
```

In Swift:

```
func casePublisher(_ publisher: SCSCasePublisherViewController,
                  fieldsToHideFromCaseFields availableFields: [String])
    -> Set<String> {

    let hideFieldSet: Set = ["MyHiddenField", "MyOtherHiddenField"]
    return hideFieldSet
}
```

You'll also need to implement the [valuesForHiddenFields](#) method, where you specify what values to use for each hidden field. This method passes you the set of hidden fields (that you specified earlier), and you'll need to provide a dictionary associating each hidden field with a value.

In Objective-C:

```
- (NSDictionary *)casePublisher:(SCSCasePublisherViewController *)publisher
    valuesForHiddenFields:(NSSet *)hiddenFields {

    NSDictionary *hideValues = @{
        @"MyHiddenField": @"The value for my hidden field.",
        @"MyHiddenField": @"Another value for hidden field."
    };

    return hideValues;
}
```

In Swift:

```
func casePublisher(_ publisher: SCSCasePublisherViewController,
    valuesForHiddenFields hiddenFields: Set<String>)
    -> [String : Any] {

    let hideValues: [String: String] = [
        "MyHiddenField" : "The value for my hidden field.",
        "MyHiddenField" : "Another value for hidden field."
    ]

    return hideValues
}
```



Note: Be sure that you specify valid values for hidden fields! If you specify an invalid value, case submission will fail and it will be unclear to the user why it happened.

If you support [authenticated users](#) and you hide fields from Case Publisher, you should also hide those fields from the Case Details view. To do this, use a similar approach and implement the [SCSCaseDetailViewControllerDelegate](#). Use the [fieldsToHideFromCaseFields](#) to specify which fields to hide.

In Objective-C:

```
- (NSSet *)caseDetail:(SCSCaseDetailViewController *)caseDetailController
    fieldsToHideFromCaseFields:(NSArray *)availableFields {

    NSSet *hideFieldsSet = [NSSet setWithObjects:@"MyHiddenField",
                                                @"MyOtherHiddenField", nil];

    return hideFieldsSet;
}
```

In Swift:

```
func caseDetail(_ caseDetailController: SCSCaseDetailViewController,
    fieldsToHideFromCaseFields availableFields: [String])
    -> Set<String> {

    let hideFieldSet: Set = ["MyHiddenField", "MyOtherHiddenField"]
    return hideFieldSet
}
```

There is no method in this delegate to specify values for hidden fields (as there is in the Case Publisher view controller delegate), because the Case Details view only shows a read-only version of these fields.

Once you've implemented your delegates, you can wire them up with one of two methods.

Method 1: Using the Default View Controllers

If you want to use the default view controllers, you'll need to find out when the view controllers are going to be shown so that you can associate your delegate implementation with the right view controller. To do this, implement `SCServiceCloudDelegate` and supply that implementation to `SCServiceCloud`.

In Objective-C:

```
[SCServiceCloud sharedInstance].delegate = self;
```

In Swift:

```
SCServiceCloud.sharedInstance().delegate = self
```

In your `SCServiceCloudDelegate` implementation, use the `willDisplayViewController` method to find out if it's a view controller you're interested in, and if so, assign the view controller delegate.

In Objective-C:

```
-(void)serviceCloud:(SCServiceCloud *)serviceCloud
    willDisplayViewController:(UIViewController *)controller
        animated:(BOOL)animated {

    if ([controller isKindOfClass:[SCSCasePublisherViewController class]]) {
        SCSCasePublisherViewController *casePublisherController =
            (SCSCasePublisherViewController *)controller;
        casePublisherController.delegate = self;
    } else if ([controller isKindOfClass:[SCSCaseDetailViewController class]]) {
        SCSCaseDetailViewController *caseDetailController =
            (SCSCaseDetailViewController *)controller;
        caseDetailController.delegate = self;
    }
}
```

In Swift:

```
func serviceCloud(_ serviceCloud: SCServiceCloud,
    willDisplay controller: UIViewController,
    animated: Bool) {


    if (controller is SCSCasePublisherViewController) {
        let publisherController = controller as! SCSCasePublisherViewController
        publisherController.delegate = self
    } else if (controller is SCSCaseDetailViewController) {
        let detailController = controller as! SCSCaseDetailViewController
        detailController.delegate = self
    }
}
```

Once you've assigned your delegates, you'll no longer see the hidden fields.

Method 2: Instantiating the View Controllers

You can also instantiate all the view controllers yourself and display them manually. You'll still need to implement the delegates using this method.

Feature	View Controller	Delegate
Case Publisher — for creating new cases	SCSCasePublisherViewController	SCSCasePublisherViewControllerDelegate
Case List — for viewing a list of a user's cases	SCSCaseListViewController	SCSCaseListViewControllerDelegate
Case Details — for inspecting the details of one case	SCSCaseDetailViewController	SCSCaseDetailViewControllerDelegate

 **Note:** If you manually display the **Case List** view controller ([SCSCaseListViewController](#)), you'll also need to manually display the **Case Detail** view controller ([SCSCaseDetailViewController](#)). When a user selects a case from the case list, present your **Case Detail** view controller from the `caseList:selectedCaseWithId:` method in your [SCSCaseListViewControllerDelegate](#) implementation. If you don't do this, nothing will happen when a user selects a specific case in the case list!

Once you've instantiated a view controller, assign your delegate using the `delegate` property on that controller. Keep in mind that you'll need to display the Case Detail view controller from the `caseList:selectedCaseWithId:` method in your [SCSCaseListViewControllerDelegate](#) implementation.

In Objective-C:

```
- (void)caseList:(SCSCaseListViewController*) caseList
    selectedCaseWithId:(NSString*) caseId {

    SCSCaseDetailViewController *controller =
        [[SCSCaseDetailViewController alloc] initWithCaseId:caseId];
    controller.delegate = self;
    [caseList.navigationController pushViewController:controller animated:YES];
}
```

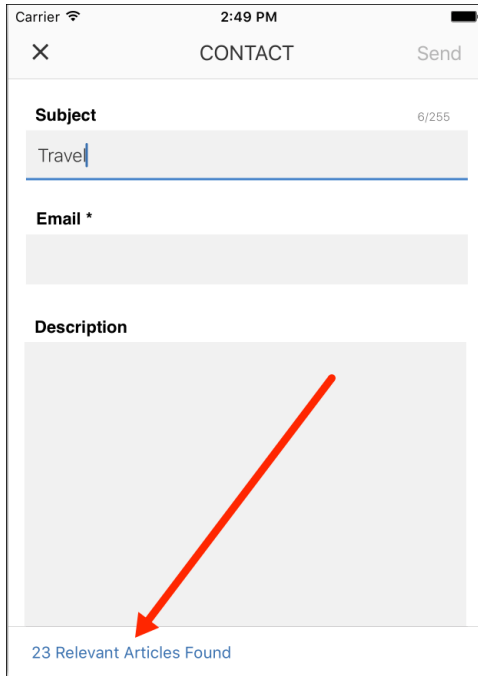
In Swift:

```
func caseList(_ caseList: SCSCaseListViewController,
              selectedCaseWithId caseId: String) {

    let controller = SCSCaseDetailViewController(caseId: caseId)
    controller.delegate = self
    caseList.navigationController!.pushViewController(controller, animated: true)
}
```

Configure Case Deflection

When a user enters information about a new case – if you have a knowledge base available to that user – the SDK automatically searches that knowledge base for relevant articles and offers them to the user. You have the ability to turn this feature on and off, as well as control which case publisher fields are used to search content.



By default, this feature is enabled and it searches the Subject and Description fields for relevant articles using those search terms. You can change this behavior using methods in the [SCSCasePublisherViewControllerDelegate](#) class. This is the delegate class for [SCSCasePublisherViewController](#). You can point the [SCSCasePublisherViewController](#) instance to your delegate implementation in one of two ways:

1. If you are displaying the Case Publisher using the default presentation (that is, using the [setInterfaceVisible:animated:completion: method](#)), refer to the **Activating Interface Using the Default Presentation** section of [Customize the Presentation for Case Management](#).
2. If you are displaying the Case Publisher by manually presenting the view controller, refer to the **Showing the View Controllers Manually** section of [Customize the Presentation for Case Management](#).

You can turn case deflection on and off with the [shouldEnableCaseDeflectionForPublisher:](#) method. You can control which fields are used for searching knowledge base articles using the [casePublisher:fieldsForCaseDeflection:](#) method.

In Objective-C:

```
- (NSSet<NSString *> *)casePublisher:(SCSCasePublisherViewController *)publisher
    fieldsForCaseDeflection:(NSArray<NSString *> *)availableFields {

    // Create the complete set of fields we want to use to search knowledge base
    NSMutableSet *deflectionSearch =
        [NSSet setWithObjects:@"CustomSubject__c",@"CustomDescription__c", nil];

    // Now build the list of fields that are confirmed to be within
    // the existing case publisher layout
    NSMutableSet *deflectionSearchFieldsInLayout = [NSMutableSet new];
    for (NSString *field in deflectionSearch) {
        if ([availableFields containsObject:field]) {
            [deflectionSearchFieldsInLayout addObject:field];
        }
    }
}
```



```

    // Return the list of fields
    return deflectionSearchFieldsInLayout;
}

```

In Swift:

```

func casePublisher(_ publisher: SCSCasePublisherViewController,
                  fieldsForCaseDeflection availableFields: [String])
    -> Set<String> {

    // Create the complete set of fields we want to use to search knowledge base
    let deflectionSearch: Set<String> = ["CustomSubject__c", "CustomDescription__c"]

    // Now build the list of fields that are confirmed to be within
    // the existing case publisher layout
    var deflectionSearchFieldsInLayout = [String]()
    for field: String in deflectionSearch {
        if (availableFields.contains(field)) {
            deflectionSearchFieldsInLayout.append(field)
        }
    }

    // Return the list of fields
    return Set(deflectionSearchFieldsInLayout)
}

```

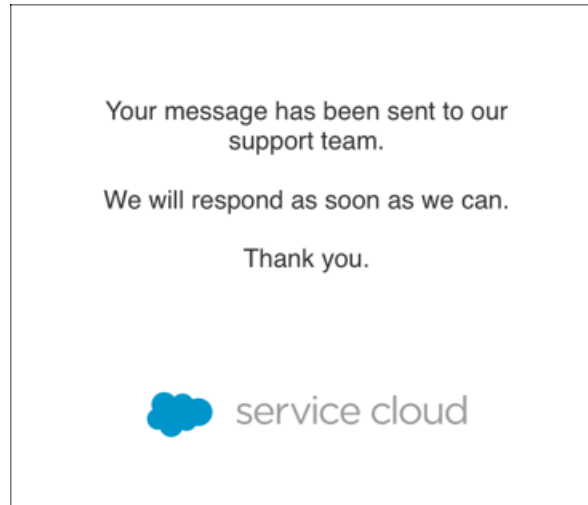


Note: Before you can use case deflection, be sure to configure your environment with a category group and root data category (using the [SCSServiceConfiguration](#) object). Also, enable the Knowledge feature (using the [knowledge.enabled](#) property). For more information, see [Quick Setup: Knowledge](#).

Customize the Case Publisher Result View

By default, when a user submits a new case from the Case Publisher screen, a standard success view appears. If you want to provide your users with more specific guidance after a case is created, one solution is to customize the view's message text and default image. If you'd like more control over what is displayed, you can present your own view by implementing the `viewForResult` method in the `SCSCasePublisherViewControllerDelegate` class.

The default Case Publisher success view contains this content:



To customize the standard message text (`ServiceCloud.CasePublisher.SuccessMessage`) and the image (`CaseSubmitSuccess`), see [Customize and Localize Strings](#) and [Customize Images](#) for instructions.

To create your own view:

1. Implement the `SCSCasePublisherViewControllerDelegate` class.

Implement the `casePublisher:viewForResult:withCaseId:error:` method where you create your custom view and return it to the Service SDK. If you don't implement the `casePublisher:viewForResult:withCaseId:error:` method (or if you return `nil`), the SDK presents the default view.

In Objective-C:


```
- (UIView *)casePublisher:(SCSCasePublisherViewController *)publisher
    viewForResult:(SCSCasePublisherResult) result
    withCaseId:(nullable NSString *)caseId
    error:(nullable NSError *)error {

    UIView* myResultView = // TO DO: Create my custom view
    return myResultView;
}
```

In Swift:

```
func casePublisher(_ publisher: SCSCasePublisherViewController,
    viewFor result: SCSCasePublisherResult,
    withCaseId caseId: String?,
    error: Error?)
    -> UIView? {

    let myResultView = // TO DO: Create my custom view
    return myResultView
}
```

 **Note:** The SDK only calls the `casePublisher:viewForResult:withCaseId:error:` method when a case submission is successful.

2. Register your delegate class with the `SCSCasePublisherViewController`.

The process for registering your delegate class is different depending on whether you're manually instantiating a Case Publisher view controller or whether you're using the default view controller.

- a. If you're manually instantiating a Case Publisher view controller, register your delegate with the [delegate](#) property of this class. For more information, see [Customize the Presentation for Case Management](#).
- b. If you're not manually instantiating the Case Publisher view controller, you can get the default view controller using the Service Cloud delegate.

Implement [SCServiceCloudDelegate](#) and supply that implementation to [SCServiceCloud](#).

In Objective-C:

```
[SCServiceCloud sharedInstance].delegate = self;
```

In Swift:

```
SCServiceCloud.sharedInstance().delegate = self
```

In your [SCServiceCloudDelegate](#) implementation, use the [willDisplayViewController](#) method to find the [SCSCasePublisherViewController](#) class and register your view controller delegate.

In Objective-C:

```
- (void)serviceCloud:(SCServiceCloud *)serviceCloud
    willDisplayViewController:(UIViewController *)controller
        animated:(BOOL)animated {

    if ([controller isKindOfClass:[SCSCasePublisherViewController class]]) {

        SCSCasePublisherViewController *casePublisherController =
            (SCSCasePublisherViewController *)controller;

        casePublisherController.delegate = // TO DO: Specify your case publisher delegate

    }
}
```

In Swift:

```
func serviceCloud(_ serviceCloud: SCServiceCloud,
    willDisplay controller: UIViewController,
    animated: Bool) {

    if controller is SCSCasePublisherViewController {

        let publisherController = controller as! SCSCasePublisherViewController

        publisherController.delegate = // TO DO: Specify your case publisher delegate

    }
}
```

After you supply your view to the Service SDK, the SDK presents it when a case is submitted.

Push Notifications for Case Activity

Using Salesforce's push notification implementation guide, it's easy to set up notifications in your app when there is activity associated with a case for an authenticated user.

You can use the [Salesforce Mobile Push Notifications Implementation Guide](#) for instructions on how to set up push notifications in your app. In short, you'll need to create a connected app for iOS, create an Apex trigger, and then package and install the components into your org.

As described in the implementation guide, an essential part of the process is to set up an Apex trigger that sends the push notification. You can use the following sample Apex code as a starting point. This code triggers a notification when an agent creates a text post.

```
trigger newCaseFeedItemPost on FeedItem (after insert) {

    for(FeedItem feedItem : Trigger.new) {

        Schema.SObjectType objectType = feedItem.parentId.getSObjectType();

        if (objectType == Case.sObjectType) {

            Case cs =
                [SELECT contactId, ownerId, caseNumber, subject FROM Case
                 WHERE id = :feedItem.parentId];

            Set<String> users = new Set<String>();

            // Determine who created or inserted this feed item
            String commentedById = feedItem.CreatedById;
            if (commentedById == null) {
                commentedById = feedItem.InsertedById;

                if (commentedById == null) {
                    commentedById = feedItem.LastEditById;
                }
            }

            // If FeedItem was created by the owner or contactId of the case,
            // then don't send a push
            if (cs.ownerId != null && !cs.ownerId.equals(commentedById)) {
                users.add(cs.ownerId);
            }

            if (cs.contactId != null && !cs.contactId.equals(commentedById)) {
                users.add(cs.contactId);
            }

            // Assemble the necessary payload parameters for Apple iOS.
            // iOS params are:
            // (<alert text>,<alert sound>,<badge count>, <free-form data>)
            // This example doesn't use badge count but does make use of free-form
            // data to pass the case ID in the notification.
            // The number of notifications that haven't been acted
            // upon by the intended recipient is best calculated
            // at the time of the push. This timing helps
            // ensure accuracy across multiple target devices.
```

```

// Create alert text
String alertText = 'New comment added to case: ' + cs.subject;


// Add the case ID so we can handle the push notification within the app
Map<String, Object> freeFormData = new Map<String, Object>();
freeFormData.put('caseid', cs.id);

// Create the payload and add it to the notification
Messaging.PushNotification msg = new Messaging.PushNotification();
Map<String, Object> payload =
    Messaging.PushNotificationPayload.apple(alertText, '', null, freeFormData);
msg.setPayload(payload);

// Send the push notification
// (First parameter needs to match your connected app name)
msg.send('ServiceCloudMobileSDK', users);
}
}
}

```


When this notification is sent to your app, you can get the SDK to handle this notification for you; it can display the Case Details or Case List screen with the relevant case information showing. To make this happen, see [Handle Remote Notifications](#).

 **Note:** If you want the SDK to handle the notification for you, your free form data **must** contain the case ID, as shown in the code snippet above. Without this information, the SDK cannot interpret the contents of the notification.

Automated Email Responses

Create automated email responses when a case is submitted from your app.

If you'd like to create an automated response when a user of your app submits a case, you can set up a **Case Auto-Response Rule**. To learn more, see the documentation on Salesforce Help: [Set Up Auto-Response Rules](#).

 **Note:** When creating an auto-response rule for guest users, be sure to add the **Web Email** field to the action layout that you've specified with the `caseCreateActionName` property. This field is used for the email response.

Once you've created the rule, a user who submits a case from your app receives an automated response.

Using Live Agent Chat

Adding the Live Agent Chat experience to your app.

[Live Agent Chat Overview](#)

Learn about the Live Agent Chat experience using the SDK.

[Quick Setup: Live Agent Chat](#)

It's easy to get started with Live Agent Chat for your iOS app. Create an `SCSChatConfiguration` object and pass it to the `startSessionWithConfiguration:` method.

[Configure a Live Agent Chat Session](#)

Before starting a Live Agent Chat session, you have several ways to optionally configure the session using the `SCSChatConfiguration` object. These configuration settings allow you to specify pre-chat fields and determine whether a session starts minimized or full screen.

[Check Live Agent Availability](#)

Before starting a session, you can check the availability of your Live Agent Chat agents and then provide your users with more accurate expectations.

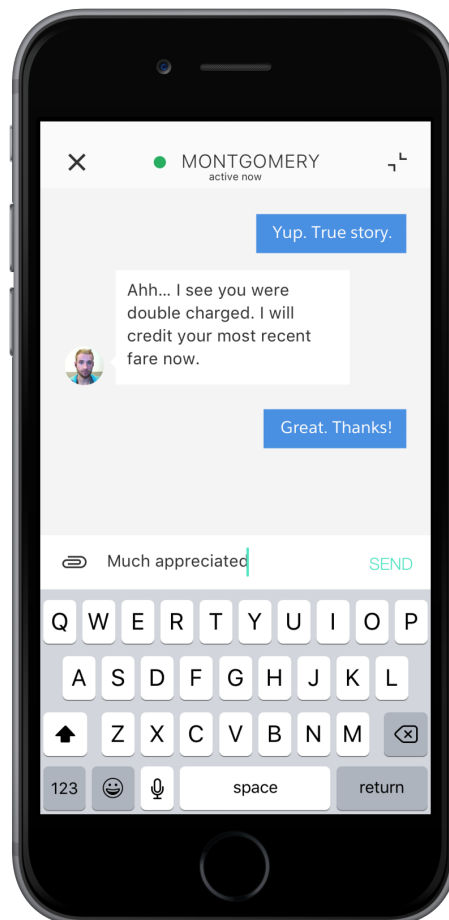
[Transfer File to Agent](#)

With Live Agent Chat, a user can transfer a file to an agent after the agent requests a file transfer.

Live Agent Chat Overview

Learn about the Live Agent Chat experience using the SDK.

With Live Agent Chat, you can provide real-time chat sessions from within your iOS app. Once you've [set up Live Agent 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.

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: Live Agent Chat

It's easy to get started with Live Agent Chat for your iOS app. Create an `SCSChatConfiguration` object and pass it to the `startSessionWithConfiguration:` method.

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

- Set up Service Cloud to work with Live Agent Chat. To learn more, see [Console Setup for Live Agent Chat](#).
- Installed the SDK. To learn more, see [Install the SDK](#).

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

1. Import the SDK. Wherever you intend to use the Live Agent Chat SDK, be sure to import the Service Common framework and the Live Agent Chat framework.

In Objective-C:

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

In Swift:

```
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 Objective-C:

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

In Swift:

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



Note: You can get the required parameters for this method from your Salesforce org. If your Salesforce Admin hasn't already set up Live Agent in Service Cloud or you need more guidance, see [Console Setup for Live Agent Chat](#).

3. (Optional) Determine whether you want Live Agent Chat to appear as a minimized view or as a full screen view.

By default, Live Agent Chat launches as a minimized view that does not take up the entire screen. This variant is advantageous when you want your user to be able to interact with the app during a chat session. Alternatively, you may want a chat session to be a full screen view that takes over their display.

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

4. (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.

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

5. (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](#) for more info.

6. To start a Live Agent Chat session, call the `startSessionWithConfiguration` method on the `chat` property.

In Objective-C:

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

In Swift:

```
SCServiceCloud.sharedInstance().chat.startSession(with: config)
```

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.

For instructions on launching the interface from a web view, see [Launch SDK from a Web View](#).

7. Listen for events and handle error conditions.

You can detect when a session ends by implementing the `chat:didEndWithReason:error:` method on the `SCSChatDelegate` delegate. Register this delegate using the `addDelegate` method on your `chat` instance. In particular, we suggest that you handle the `SCSChatEndReasonAgent` reason (for when an agent ends a session) as well as the `SCSChatNoAgentsAvailableError` error code (for when there are no agents available).



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.



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

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 `SCSChatDelegate` 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 `chat:didEndWithReason:error:` method and show a dialog when appropriate.

In Objective-C:

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

@interface ViewController : UIViewController <SCSChatDelegate>

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];
```



```

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

- (IBAction)startChat:(id)sender {

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

    // Start the session
    [[SCSServiceCloud sharedInstance].chat startSessionWithConfiguration:config];
}

- (void)chat:(SCSSChat *)chat didEndWithReason:(SCSSChatEndReason)reason
    error:(NSError *)error {

    NSString *description = nil;

    // Here we'll handle the situation where the agent ends the session
    // and when there are no agents available...
    if (reason == SCSSChatEndReasonAgent) {
        description = @"The agent has ended the session.";
    } else if (reason == SCSSChatEndReasonSessionError &&
        error.code == SCSSChatNoAgentsAvailableError) {
        description = @"It looks like there are no agents available. Try again later.";
    }

    if (description != nil) {
        UIAlertController *alert = [UIAlertController
            alertControllerWithTitle:@"Session Ended"
            message:description
            preferredStyle:UIAlertControllerStyleAlert];

        UIAlertAction* okAction = [UIAlertAction
            actionWithTitle:@"OK"
            style:UIAlertActionStyleDefault
            handler:^(UIAlertAction * action)
            {
                NSLog(@"OK action");
            }];

        [alert addAction:okAction];
        [self presentViewController:alert animated:YES completion:nil];
    }
}
@end

```

In Swift:

```

import UIKit
import ServiceCore

```

```

import ServiceChat

class ViewController : UIViewController, SCSCChatDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Add our chat delegate
        SCSServiceCloud.sharedInstance().chat.add(self)
    }

    @IBAction func startChat(_ sender: AnyObject) {

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

        // Start a session
        SCSServiceCloud.sharedInstance().chat.startSession(with: config)
    }

    func chat(_ chat: SCSCChat!, didEndWith reason: SCSCChatEndReason,
              error: Error!) {

        var description = ""

        // Here we'll handle the situation where the agent ends the session
        // and when there are no agents available...
        if (reason == .agent) {
            description = "The agent has ended the session."
        } else if (reason == .sessionError &&
                   (error as NSError).code == SCSCChatErrorCode.noAgentsAvailableError.rawValue) {
            description = "It looks like there are no agents available. Try again later."
        }

        if (description != "") {
            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)
        }
    }
}

```

Configure a Live Agent Chat Session

Before starting a Live Agent Chat session, you have several ways to optionally configure the session using the `SCSChatConfiguration` object. These configuration settings allow you to specify pre-chat fields and determine whether a session starts minimized or full screen.

When you start a Live Agent Chat session, you specify an `SCSChatConfiguration` object as one of the arguments. This object contains all the configuration settings necessary for Live Agent Chat to start a session. To create an `SCSChatConfiguration` object, you specify information about your org and deployment (as described in [Quick Setup: Live Agent Chat](#)), and that is all that is required. However, there are other options you can set using `SCSChatConfiguration`.

 **Note:** Be sure not to start a Live Agent Chat session until you've fully configured the `SCSChatConfiguration` object.

The following features are available for configuration:

Method Name	Description	Type : Default Value
<code>presentationStyle</code>	<p>By default, Live Agent Chat launches as a minimized view that does not take up the entire screen. This variant is advantageous when you want your user to be able to interact with the app during a chat session. Alternatively, you may want a chat session to be a full screen view that takes over their display.</p> <p>To configure the presentation style, call the <code>presentationStyle</code> method on the <code>SCSChatConfiguration</code> object. Specify <code>SCSChatPresentationStyleNonBlocking</code> for the minimized view; specify <code>SCSChatPresentationStyleFullscreen</code> to initialize in the full-screen chat experience.</p> <p>The user can always switch between these states during a chat session — when in full-screen mode, the user taps the minimize button to minimize the session; when in minimized mode, the user taps the chat thumbnail to go into full-screen mode</p>	<p><code>SCSChatPresentationStyle</code> enum :</p> <p><code>SCSChatPresentationStyleNonBlocking</code></p>
<code>prechatFields</code>	<p>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.</p> <p>Specify each field as an item in an array of <code>SCSPrechatObject</code> objects and pass the array to the <code>prechatFields</code> method on the <code>SCSChatConfiguration</code> object.</p> <p>There are three types of pre-chat fields:</p> <ul style="list-style-type: none"> • <code>SCSPrechatObject</code> does not require user input. • <code>SCSPrechatTextInputObject</code> (a subclass of <code>SCSPrechatObject</code>) can take user input. • <code>SCSPrechatPickerObject</code> (a subclass of <code>SCSPrechatObject</code>) provides the user with a dropdown list of options. 	<p><code>SCSPrechatObject</code> array: nil</p>

Method Name	Description	Type : Default Value
	<p>You initialize a pre-chat object with a string used for the pre-chat label.</p> <p>When using a <code>SCSPrechatTextInputObject</code>, you can control several other characteristics:</p> <ul style="list-style-type: none"> • <code>required</code>— to specify whether the field is required. • <code>keyboardType</code>— to use other standard keyboards (such as <code>UIKeyboardTypeEmailAddress</code>). • <code>autocapitalizationType</code>— to control how text capitalization works. • <code>autocorrectionType</code>— to control auto-correction. • <code>maxLength</code>— to specify the maximum length of the field. <p>When using a <code>SCSPrechatPickerObject</code>, you can access two properties:</p> <ul style="list-style-type: none"> • <code>required</code>— to specify whether the field is required. • <code>options</code> — which is an array of <code>SCSPrechatPickerOption</code> objects. Each <code>SCSPrechatPickerOption</code> object is one item in the dropdown list. 	



Example: The following example starts a session in full-screen mode with a few pre-chat fields: one required text field for email, one optional text field, one optional dropdown field, and one hidden custom field.

In Objective-C:

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

// Set full screen mode
config.presentationStyle = SCSChatPresentationStyleFullscreen;

// 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];

// Add an optional name field (with no auto-correction and word capitalization)
SCSPrechatTextInputObject* nameField = [[SCSPrechatTextInputObject alloc]
                                          initWithLabel:@"Name"];
nameField.required = NO;
```

```

nameField.autocorrectionType = UITextAutocorrectionTypeNo;
nameField.autocapitalizationType = UITextAutocapitalizationTypeWords;
[config.prechatFields addObject:nameField];

// Add a dropdown field for department name
NSMutableArray<SCSPrechatPickerOption *> *pickerOptions = [NSMutableArray new];
[pickerOptions addObject:[SCSPrechatPickerOption alloc] initWithLabel:@"Sales"
                                                                value:@"sales"];
[pickerOptions addObject:[SCSPrechatPickerOption alloc] initWithLabel:@"Billing"
                                                                value:@"billing"];
SCSPrechatPickerObject* pickerObject = [[SCSPrechatPickerObject alloc]
initWithLabel:@"Department"
options:pickerOptions]];
pickerObject.required = NO;
[config.prechatFields addObject:pickerObject];

// Add custom data directly to agent
SCSPrechatObject* customData = [[SCSPrechatObject alloc]
                                initWithLabel:@"DataForAgent"
                                value:@"CUSTOM-DATA-HERE"];
[config.prechatFields addObject:customData];

// Start session!
[[SCServiceCloud sharedInstance].chat startSessionWithConfiguration:config];

```

In Swift:

```

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

// Set full screen mode
config?.presentationStyle = .fullscreen

// 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.add(emailField)

// Add an optional name field (with no auto-correction and word capitalization)
let nameField = SCSPrechatTextInputObject(label: "Name")
nameField?.isRequired = false
nameField?.autocorrectionType = .no
nameField?.autocapitalizationType = .words
config?.prechatFields.add(nameField)

// Add a dropdown field for department name
let pickerOptions = NSMutableArray()
pickerOptions.add(SCSPrechatPickerOption(label:"Sales", value:"sales"))
pickerOptions.add(SCSPrechatPickerOption(label:"Billing", value:"billing"))
let pickerObject = SCSPrechatPickerObject(label: "Department",

```

```

options: pickerOptions as NSArray as!
[SCSPrechatPickerOption])
pickerObject?.isRequired = false
config?.prechatFields.add(pickerObject)

// Add custom data directly to agent
let customData = SCSPrechatObject(label: "DataForAgent", value: "CUSTOM-DATA-HERE")
config?.prechatFields.add(customData)

// Start session!
SCSServiceCloud.sharedInstance().chat.startSession(with: config)

```

Check Live Agent Availability

Before starting a session, you can check the availability of your Live Agent Chat agents and then provide your users with more accurate expectations.

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

In Objective-C:

```

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

[[SCSServiceCloud sharedInstance].chat
    determineAvailabilityWithConfiguration:config
                                completion:^(NSError *error, BOOL available)
{
    if (error != nil) {
        // Handle error
    }
    else if (available) {
        // Enable chat button
    }
    else {
        // Disable button or warn user that no agents are available
    }
});

```

In Swift:

```

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

SCSServiceCloud.sharedInstance().chat.determineAvailability(with: config,
                                                         completion: { (error: Error?, available: Bool) in

```

```

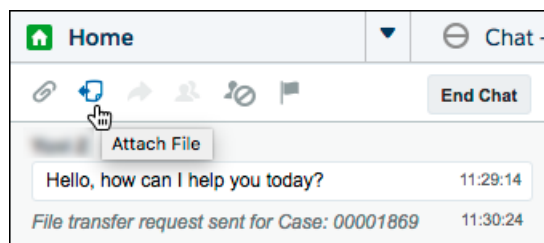
if (error != nil) {
    // Handle error
}
else if (available) {
    // Enable chat button
}
else {
    // Disable button or warn user that no agents are available
}
})

```

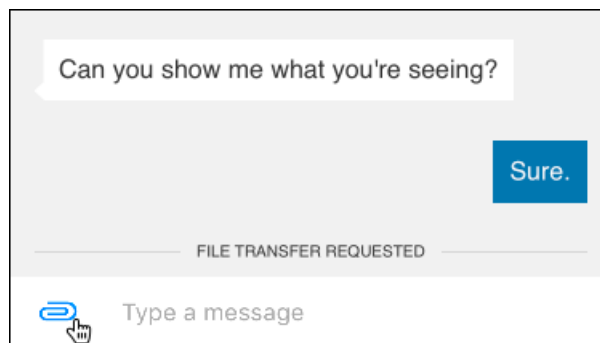
Transfer File to Agent

With Live Agent Chat, a user can transfer a file to an agent after the agent requests a file transfer.

In addition to text-based conversations, an 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 within the app, and can then send a file using the paperclip button.



See [Transfer Files During a Chat](#) in Salesforce Help for details about setting up this functionality in Service Cloud. No coding is necessary in your app to make this behavior work.

Using SOS

Adding the SOS experience to your app.

[SOS Overview](#)

Learn about the SOS experience using the SDK.

[SOS Example App](#)

The SOS example app demonstrates many of the SOS features and can help you get up to speed.

[Quick Setup: SOS](#)

It's easy to get started with SOS for your iOS app. Use `SOSSessionManager` to start an SOS session.

[Configure an SOS Session](#)

Before starting an SOS session, you can optionally configure the session using the `SOSOptions` object. These configuration settings allow you to enable or disable cameras, determine what screen a session starts on, specify whether network tests are performed, and control other features.

[Two-Way Video](#)

In addition to screen sharing, the SOS SDK lets your customer share their device's live camera feed with an agent. The customer's front-facing camera allows for a video conversation with an agent. The back-facing camera provides a great way for a customer to show something to an agent, rather than have to explain it.

[Listen to Events and Handle Errors](#)

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

[Check SOS Agent Availability](#)

Before starting a session, you can check the availability of your SOS agents and then provide your users with more accurate expectations.

[Enable and Disable Screen Sharing](#)

There are some scenarios where you may want to programmatically turn off screen sharing in mid-session. You can enable and disable screen sharing using the `screenSharing` property.

[Field Masking](#)

If an application contains sensitive information that an agent shouldn't see during an SOS session, you can hide this information from the agent.

[Custom Data](#)

Use custom data to identify customers, send error messages, issue descriptions, or identify the page the SOS session was initiated from.

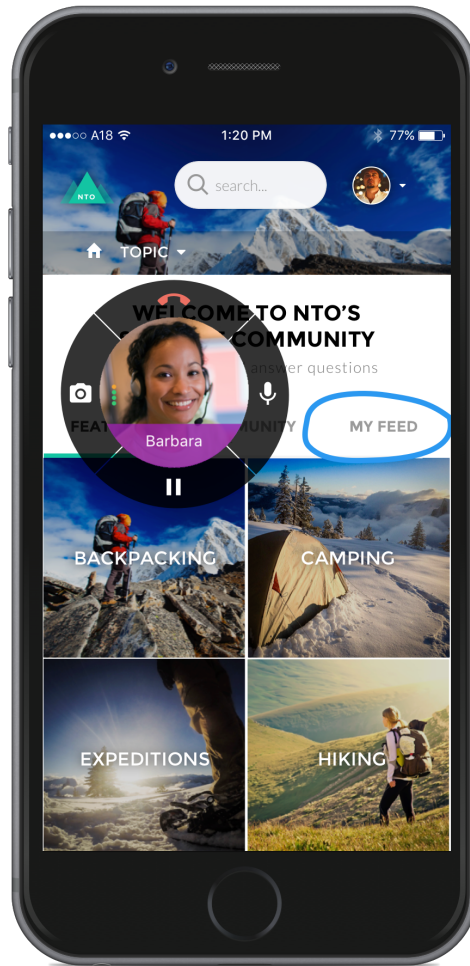
[Replace the SOS UI](#)

If you'd like to customize the SOS UI, you can create your own UI by subclassing the `UIViewController` class associated with that phase of the SOS session.

SOS Overview

Learn about the SOS experience using the SDK.

SOS lets you easily add real-time video and screen sharing support to your native iOS app. Once you've [set up Service Cloud for SOS](#), it takes [just a few calls to the SDK](#) to have your app ready to handle agent calls and to support screen sharing. With screen sharing, agents can even make annotations directly on the customer's screen.



And with just a few more [configuration](#) changes, you can provide [two-way video support](#) from your app. This functionality can include front-facing camera support, back-facing camera support, or both.



There are several other ways you can set up your SOS environment, including [masking sensitive fields](#) and [passing custom data back to your org](#).

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.

Check out [Listen to Events and Handle Errors](#) for information about how to handle event changes. In particular, you'll want to listen for error conditions and present alerts to the user when applicable.

Let's get started.

SOS Example App

The SOS example app demonstrates many of the SOS features and can help you get up to speed.

Before using the example app, you must:

- Set up Service Cloud to work with SOS. To learn more, see [Console Setup for SOS](#).
- Install [CocoaPods](#), which is needed for the project's dependencies.

Once you've reviewed these prerequisites, you're ready to download and use the SOS example app.

1. Clone the git repository where the example project is located: <https://github.com/goinstant/ios-sdk-guides>.

```
git clone https://github.com/goinstant/ios-sdk-guides
```

2. Change directories into the example folder.

```
cd ios-sdk-guides
```

3. Run the CocoaPods installer.

```
pod install
```

This command installs all the dependencies required for this project.

4. Launch the project workspace in Xcode.

```
open SOSExamples.xcworkspace
```

When using CocoaPods, you must be sure to launch the workspace file (`.xcworkspace`) and NOT the project file (`.xcodeproj`). `SOSExamples.xcodeproj` will not work on its own.

5. Open the `SOSSettings.plist` file and update it with your org's setup values (Salesforce Organization ID, Deployment ID, Live Agent Pod). If the settings are not valid, an error occurs when you start an SOS session.

To learn more about setting up your org, see [Console Setup for SOS](#).

6. Build and run the `SOSExamples` project!

This sample application shows how you can approach integrating SOS into your own app. This app is almost entirely wired together using the storyboard.

If you run into network issues while connecting with an agent, see [SOS Network Troubleshooting Guide](#).

Quick Setup: SOS

It's easy to get started with SOS for your iOS app. Use `SOSSessionManager` to start an SOS session.

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

- Set up Service Cloud to work with SOS. To learn more, see [Console Setup for SOS](#).
- Installed the SDK. To learn more, see [Install the SDK](#).

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

1. Import the SDK. Wherever you intend to use the SOS SDK, be sure to import the Service Common framework and the SOS framework.

In Objective-C:

```
@import ServiceCore;
#import ServiceSOS;
```

In Swift:

```
import ServiceCore
import ServiceSOS
```

2. Create an `SOSOptions` object with information about your LiveAgent pod, your Salesforce org ID, and the deployment ID.

In Objective-C:

```
SOSOptions *options = [SOSOptions optionsWithLiveAgentPod:@"YOUR-POD-NAME"
                                           orgId:@"YOUR-ORG-ID"
                                           deploymentId:@"YOUR-DEPLOYMENT-ID"];
```

In Swift:

```
let options = SOSOptions(liveAgentPod: "YOUR-POD-NAME",
                        orgId: "YOUR-ORG-ID",
                        deploymentId: "YOUR-DEPLOYMENT-ID")
```



Note: You can get the required parameters for this method from your Salesforce org. If your Salesforce Admin hasn't already set up SOS in Service Cloud or you need more guidance, see [Console Setup for SOS](#).

3. (Optional) Specify additional configuration settings before starting a session.

Before starting an SOS session, you can optionally configure the session using the [SOSOptions](#) object. These configuration settings allow you to enable or disable cameras, determine what screen a session starts on, specify whether network tests are performed, and control other features. To learn more, see [Configure an SOS Session](#).

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](#) for more info.

5. To start an SOS session, call [startSessionWithOptions](#) on the [SOSSessionManager](#) shared instance.

You can start a session when the view controller loads, or from a UI action.

In Objective-C:

```
[[SCServiceCloud sharedInstance].sos startSessionWithOptions:options];
```

In Swift:

```
SCServiceCloud.sharedInstance().sos.startSession(with: options)
```

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.

For instructions on launching the interface from a web view, see [Launch SDK from a Web View](#).

6. Listen for events and handle error conditions.

You can listen for state changes that occur during a session life cycle by implementing [SOSDelegate](#) methods. Register this delegate using the [addDelegate](#) method on your [SOS instance](#). In particular, we suggest that you implement the [sosDidStop:reason:error:](#) method to handle session termination. To learn more, see [Listen to Events and Handle Errors](#).



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.

7. (Optional) If you want to programmatically stop a session, call the [stopSession](#) method on the [SOSSessionManager](#) shared instance.

In Objective-C:

```
[[SCServiceCloud sharedInstance].sos stopSession];
```

In Swift:

```
SCServiceCloud.sharedInstance().sos.stopSession()
```

Alternatively, you can call the [stopSession](#) method with a completion block.

For additional details on customizing the SOS experience in your app, see the other topics covered in [Using SOS](#). If you run into network issues while connecting with an agent, see [SOS Network Troubleshooting Guide](#).



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

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

- Implement the `SOSDelegate` protocol so that you can be notified when there are errors or state changes.
- Specify `self` as an SOS delegate.
- Start an SOS session in the button action.
- Implement the `sosDidStop:reason:error:` method and show a dialog when appropriate.

In Objective-C:

```
#import <UIKit/UIKit.h>
#import ServiceCore;
#import ServiceSOS;

@interface ViewController : UIViewController <SOSDelegate>

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    // Add our SOS delegate
    [[SCServiceCloud sharedInstance].sos addDelegate:self];
}

- (IBAction)startSOS:(id)sender {

    // Create options object
    SOSOptions *options = [SOSOptions optionsWithLiveAgentPod:@"YOUR-POD-NAME"
                                                orgId:@"YOUR-ORG-ID"
                                                deploymentId:@"YOUR-DEPLOYMENT-ID"];

    // Start the session
    [[SCServiceCloud sharedInstance].sos startSessionWithOptions:options];
}

// Delegate method for session stop event.
// You can also check for fatal errors with this delegate method.
- (void)sos:(SOSSessionManager *)sos didStopWithReason:(SOSStopReason)reason
    error:(NSError *)error {

    NSString *title = nil;
    NSString *description = nil;

    // If there's an error...
    if (error != nil) {

        switch (error.code) {
```

```

// No agents available
case SOSNoAgentsAvailableError: {
    title = @"Session Failed";
    description = @"It looks like there are no agents available. Try again later.";

    break;
}

// Network test failure
case SOSNetworkTestError: {
    title = @"Session Failed";
    description = @"Insufficient network. Check network quality and try again.";
    break;
}

// TO DO: Use SOSErrorCode to check for ALL other error conditions
//         in order to give a more clear explanation of the error.
default: {
    title = @"Session Error";
    description = @"Unknown session error.";
    break;
}
}

// Else if session stopped without an error condition...
} else {

    switch (reason) {

        // Handle the agent disconnect scenario
        case SOSStopReasonAgentDisconnected: {
            title = @"Session Ended";
            description = @"The agent has ended the session.";
            break;
        }

        // TO DO: Use SOSStopReason to check for
        //         other reasons for session ending...
        default: {
            break;
        }
    }
}

// Display dialog if we have something to say...
if (title != nil) {
    UIAlertController *alert = [UIAlertController
                                alertControllerWithTitle:title
                                message:description
                                preferredStyle:UIAlertControllerStyleAlert];

    UIAlertAction* okAction = [UIAlertAction
                                actionWithTitle:@"OK"
                                style:UIAlertActionStyleDefault

```

```

        handler:^(UIAlertAction * action)
        {
            NSLog(@"OK action");
        }];

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

@end

```

In Swift:

```

import UIKit
import ServiceCore
import ServiceSOS

class ViewController: UIViewController, SOSDelegate {

    override func viewDidLoad() {
        super.viewDidLoad()

        // Add our SOS delegate
        SCServiceCloud.sharedInstance().sos.add(self)
    }

    @IBAction func startSOS(sender: AnyObject) {

        // Create options object
        let options = SOSOptions(liveAgentPod: "YOUR-POD-NAME",
                                orgId: "YOUR-ORG-ID",
                                deploymentId: "YOUR-DEPLOYMENT-ID")

        // Start the session
        SCServiceCloud.sharedInstance().sos.startSessionWithOptions(options)
    }

    // Delegate method for session stop event.
    // You can also check for fatal errors with this delegate method.
    func sos(_ sos: SOSSessionManager!, didStopWith reason: SOSStopReason,
            error: Error!) {

        var title = ""
        var description = ""

        // If there's an error...
        if (error != nil) {

            switch (error as NSError).code {

                // No agents available
                case SOSErrorCode.SOSNoAgentsAvailableError.rawValue:
                    title = "Session Failed"

```

```

        description = "It looks like there are no agents available. Try again later."

// Insufficient network error
case SOSErrorCode.SOSInsufficientNetworkError.rawValue:
    title = "Session Failed"
    description = "Insufficient network. Check network quality and try again."

// TO DO: Use SOSErrorCode to check for ALL other error conditions
//         in order to give a more clear explanation of the error.
default:
    title = "Session Error"
    description = "Unknown session error."
}

// Else if session stopped without an error condition...
} else {

    switch reason {

        // Handle the agent disconnect scenario
        case .agentDisconnected:
            title = "Session Ended"
            description = "The agent has ended the session."

        // TO DO: Use SOSStopReason to check for
        //         other reasons for session ending...
        default:
            break
    }
}

// Display dialog if we have something to say...
if (title != "") {

    let alert = UIAlertController(title: title,
                                message: description,
                                preferredStyle: .alert)

    let okAction = UIAlertAction(title: "OK",
                                style: .default,
                                handler: nil)

    alert.addAction(okAction)
    self.present(alert, animated: true, completion: nil)
}
}
}

```

Configure an SOS Session

Before starting an SOS session, you can optionally configure the session using the [SOSOptions](#) object. These configuration settings allow you to enable or disable cameras, determine what screen a session starts on, specify whether network tests are performed, and control other features.

When you start an SOS session using `SOSSessionManager`, you specify an `SOSOptions` object as one of the arguments. This object contains all the configuration settings necessary for SOS to start a session. To create an `SOSOptions` object, you specify information about your org and deployment (as described in [Quick Setup: SOS](#)), and that is all that is required. However, there are many other options you can set using `SOSOptions`.



Note: Be sure not to start an SOS session until you've fully configured the `SOSOptions` object.

The following features are available for configuration:

Feature	Description	Type & Default Value
<code>customFieldData</code>	Dictionary that can be used to send custom data to your Salesforce org. See Custom Data .	<code>NSMutableDictionary</code> — Default: <code>nil</code>
<code>featureAgentVideoStreamEnabled</code>	Whether the agent video stream is enabled for the session.	<code>BOOL</code> — Default: <code>YES/true</code>
<code>featureClientBackCameraEnabled</code>	Whether the back-facing camera is enabled for the session. See Two-Way Video .	<code>BOOL</code> — Default: <code>NO/false</code>
<code>featureClientFrontCameraEnabled</code>	Whether the front-facing (selfie) camera is enabled for the session. See Two-Way Video .	<code>BOOL</code> — Default: <code>NO/false</code>
<code>featureClientScreenSharingEnabled</code>	Whether screen sharing is enabled for the session.	<code>BOOL</code> — Default: <code>YES/true</code>
<code>featureNetworkTestEnabled</code>	Whether the network test is enabled before and during a session.	<code>BOOL</code> — Default: <code>YES/true</code>
<code>initialAgentVideoStreamActive</code>	Whether the agent video stream is active when starting a session.	<code>BOOL</code> — Default: <code>YES/true</code>
<code>initialCameraType</code>	The initial view (screen sharing, front-facing camera, or back-facing camera) when starting a session.	<code>SOSCameraType</code> enumerated type — Default: <code>screenSharing</code>
<code>remoteLoggingEnabled</code>	Determines whether session logs are sent for collection. Logs sent remotely do not collect personal information. Unique IDs are created for tying logs to sessions and those IDs cannot be correlated back to specific users.	<code>BOOL</code> — Default: <code>YES/true</code>
<code>viewControllerClass</code>	Lets you override the SOS UI. See Replace the SOS UI .	<code>SOSUIPhase</code> enumerated type.
<code>sessionRetryTime</code>	The length of time (in seconds) before SOS prompts the user to retry or cancel.	<code>NSTimeInterval</code> — Default: 30

As you can see from the table, by default, an SOS session starts in screen sharing mode, with both cameras disabled. If you don't want default values, manually change the options before starting a session.



Example: The following example starts a session with the back-facing camera showing, the front-facing camera enabled, and screen sharing disabled.

In Objective-C:

```
SOSOptions *options = [SOSOptions optionsWithLiveAgentPod:@"YOUR-POD-NAME"
                                     orgId:@"YOUR-ORG-ID"
                                     deploymentId:@"YOUR-DEPLOYMENT-ID"];

[options setFeatureClientBackCameraEnabled: YES];
[options setFeatureClientFrontCameraEnabled: YES];
[options setFeatureClientScreenSharingEnabled: NO];
[options setInitialCameraType: SOSCameraTypeBackFacing];

[[SCServiceCloud sharedInstance].sos startSessionWithOptions:options];
```

In Swift:

```
let options = SOSOptions(liveAgentPod: "YOUR-POD-NAME",
                        orgId: "YOUR-ORG-ID",
                        deploymentId: "YOUR-DEPLOYMENT-ID")

options!.featureClientBackCameraEnabled = true
options!.featureClientFrontCameraEnabled = true
options!.featureClientScreenSharingEnabled = false
options!.initialCameraType = .backFacing

SCServiceCloud.sharedInstance().sos.startSession(with: options)
```

Two-Way Video

In addition to screen sharing, the SOS SDK lets your customer share their device's live camera feed with an agent. The customer's front-facing camera allows for a video conversation with an agent. The back-facing camera provides a great way for a customer to show something to an agent, rather than have to explain it.

By default, after a connection is established, the camera shows the agent in the full-screen view and the customer's camera in the picture-in-picture view. If a device has both a front-facing and back-facing camera, the customer can swap cameras by double-tapping the screen during the two-way video session. The customer can also tap the picture-in-picture view to swap the full-screen view with the picture-in-picture view.



You can programmatically configure which cameras are available and which camera the session starts with.

[Configure Two-Way Video](#)

Two-way video for SOS is disabled by default. You can enable it by accessing one or both cameras in the user's device with the `SOSOptions` object used when starting the SOS session. You can also configure the initial camera view when the session starts.

Configure Two-Way Video

Two-way video for SOS is disabled by default. You can enable it by accessing one or both cameras in the user's device with the `SOSOptions` object used when starting the SOS session. You can also configure the initial camera view when the session starts.



Note: Always test two-way video on an actual device, rather than using the simulator. The Xcode simulator doesn't have access to a camera, so it doesn't provide you with an accurate experience.

1. Create an `SOSOptions` object using the `optionsWithLiveAgentPod:orgId:deploymentId:` method, as described in [Quick Setup: SOS](#).

2. Initialize access to one or both of the cameras on the user's device by calling `setFeatureClientFrontCameraEnabled` and `setFeatureClientBackCameraEnabled`.

In Objective-C:

```
[options setFeatureClientFrontCameraEnabled: YES];  
[options setFeatureClientBackCameraEnabled: YES];
```

In Swift:

```
options.featureClientFrontCameraEnabled = true  
options.featureClientBackCameraEnabled = true
```

To learn more, see [Configure an SOS Session](#).

3. Determine what is displayed when a session starts.

By default, a session starts in screen sharing mode. Alternatively, you can start a session using one of the cameras. For example, the following command starts a session using the back-facing camera. In Objective-C:

```
[options setInitialCameraType: SOSCameraTypeBackFacing];
```

In Swift:

```
options.initialCameraType = .backFacing
```

If you'd rather start a session with the front-facing camera, use this command instead. In Objective-C:

```
[options setInitialCameraType: SOSCameraTypeFrontFacing];
```

In Swift:

```
options.initialCameraType = .frontFacing
```

To learn more, see [Configure an SOS Session](#).

4. Determine whether you want to allow the screen sharing feature.

Even if you start a session with the camera, the screen sharing function is still enabled by default. This functionality may be appropriate for your use case. However, you can disable screen sharing altogether with the following call. In Objective-C:

```
[options setFeatureClientScreenSharingEnabled: NO];
```

In Swift:

```
options.featureClientScreenSharingEnabled = false
```

To learn more, see [Configure an SOS Session](#).

5. Start an SOS session.

In Objective-C:

```
[[SCServiceCloud sharedInstance].sos startSessionWithOptions:options];
```

In Swift:

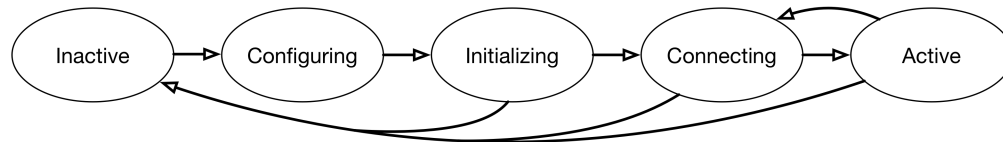
```
SCServiceCloud.sharedInstance().sos.startSession(with: options)
```

To learn more, see [Quick Setup: SOS](#).

Listen to Events and Handle Errors

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

The `SOSSessionManager singleton` maintains all information related to SOS sessions. One of its properties (`state`) is an `SOSSessionState` object, which maintains the current state of SOS. This state object can be in one of five different states:



Inactive

No active session; no outgoing/incoming SOS traffic

Configuring

Performing a pre-initialization configuration step, such as network testing

Initializing

Preparing to connect

Connecting

Attempting a connection to a live agent

Active

Connected with agent; session is fully established

Throughout a session, your application might want information related to the session state. You can monitor state changes by implementing `SOSDelegate`. Use the `addDelegate` on the `SOS instance` to register your delegate.

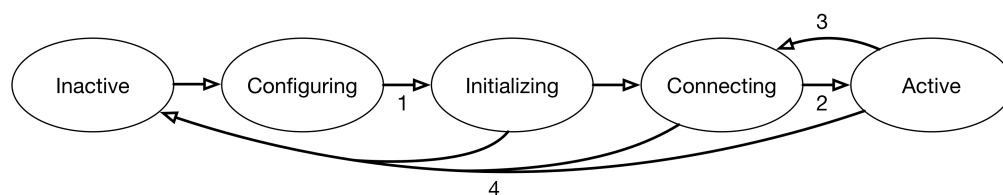
Listening to State Changes

If you want to know every time an SOS session state changes, the `sos:stateDidChange:previous:` method is called for every change.

A more specific set of delegate methods are available to track specific state changes:

1. `sosDidStart:` — invoked when a session is possible and the user has agreed to start a session.
2. `sosDidConnect:` — invoked when the SOS session has connected.
3. `sosWillReconnect:` — invoked if network connectivity issues arise, and the session will attempt to reconnect.
4. `sosDidStop:reason:error:` — invoked when the session has ended.

These state changes are labeled with an associated number in the diagram below.



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 `SOSDelegate` methods for this purpose:

1. As mentioned in the previous section, `sosDidStop:reason:error:` is the method to use for session termination. Inspect the reason (`SOSStopReason`) to determine why the session stopped. Typically, the session stops due to a normal event (for example, `SOSStopReasonUserDisconnected` or `SOSStopReasonAgentDisconnected`). If the reason is `SOSStopReasonSessionError`, check the `error` parameter for more detail and compare the error code to `SOSErrorCode` values. For instance, when there are no agents available to take a call, the error is `SOSNoAgentsAvailableError`.
2. You can track all SOS errors (including the session errors that are passed to `sosDidStop:reason:error:`) with the `sos:didError:` method. Compare the error code to `SOSErrorCode` to determine what kind of error occurred.

See the sample code below for a basic implementation of `sosDidStop:reason:error:` and `sos:didError:`.



Example: Sample Code in SOS Test App

For a detailed example that implements `SOSDelegate`, refer to the SOS example app: [SOS Example App](#). In particular, look at the AppDelegate class: <https://github.com/goinstant/ios-sdk-guides/blob/master/SOSExamples/AppDelegate.m>.



Example: Basic `SOSDelegate` Example

This sample code does the following:

- Implements the `SOSDelegate` protocol.
- Implements the `sos:stateDidChange:previous:` method.
- Implements the `sosDidConnect:` method.
- Implements the `sosDidStop:reason:error:` method and includes some error handling logic.
- Implements the `sos:didError:` method.

In Objective-C:

```
#import <UIKit/UIKit.h>
#import ServiceCore;
#import ServiceSOS;

@interface MySOSDelegateImplementation : NSObject <SOSDelegate>

@end

@implementation MyDelegateImplementation

// TO DO: Register this delegate using [[SCServiceCloud sharedInstance].sos
addDelegate:self]

// Delegate method for state change.
- (void)sos:(SOSSessionManager *)sos stateDidChange:(SOSSessionState)current
                                     previous:(SOSSessionState)previous {
    NSLog(@"SOS state changed...");
    if (current == SOSSessionStateConnecting) {
        NSLog(@"SOS now connecting...");
    }
}

// Delegate method for connect state.
```

```

- (void)sosDidConnect:(SOSSessionManager *)sos {
    NSLog(@"SOS session connected...");
}

// Delegate method for session stop event.
// You can also check for fatal errors with this delegate method.
- (void)sos:(SOSSessionManager *)sos didStopWithReason:(SOSStopReason)reason
    error:(NSError *)error {

    NSString *title = nil;
    NSString *description = nil;

    // If there's an error...
    if (error != nil) {

        switch (error.code) {

            // No agents available
            case SOSNoAgentsAvailableError: {
                title = @"Session Failed";
                description = @"It looks like there are no agents available. Try again later.";

                break;
            }

            // Network test failure
            case SOSNetworkTestError: {
                title = @"Session Failed";
                description = @"Insufficient network. Check network quality and try again.";
                break;
            }

            // TO DO: Use SOSErrorCode to check for ALL other error conditions
            //           in order to give a more clear explanation of the error.
            default: {
                title = @"Session Error";
                description = @"Unknown session error.";
                break;
            }
        }
    }

    // Else if session stopped without an error condition...
    else {

        switch (reason) {

            // Handle the agent disconnect scenario
            case SOSStopReasonAgentDisconnected: {
                title = @"Session Ended";
                description = @"The agent has ended the session.";
                break;
            }

            // TO DO: Use SOSStopReason to check for

```

```

        //          other reasons for session ending...
        default: {
            break;
        }
    }
}

// Do we have an error to report?
if (title != nil) {
    // TO DO: Display an alert using title & description
    NSLog(@"\nSOS ALERT Title: %@\nDescription: %@", title, description);
}
}

- (void)sos:(SOSSessionManager *)sos didError:(NSError *)error {
    NSLog(@"SOS error (%d): %@", error.code, error.localizedDescription);
}

@end

```

In Swift:

```

import UIKit
import ServiceCore
import ServiceSOS

class MySOSDelegateImplementation: NSObject, SOSDelegate {

    // TO DO: Register this delegate using
    SCServiceCloud.sharedInstance().sos.addDelegate(self)

    // Delegate method for state change.
    func sos(_ sos: SOSSessionManager!, stateDidChange current: SOSSessionState,
            previous: SOSSessionState) {

        NSLog("SOS state changed...")

        if (current == .connecting) {
            NSLog("SOS now connecting...")
        }
    }

    // Delegate method for connect state.
    func sosDidConnect(_ sos: SOSSessionManager!) {
        NSLog("SOS session connected...")
    }

    // Delegate method for session stop event.
    // You can also check for fatal errors with this delegate method.
    func sos(_ sos: SOSSessionManager!, didStopWith reason: SOSStopReason,
            error: Error!) {

        var title = ""
        var description = ""
    }
}

```



```

// If there's an error...
if (error != nil) {

    switch (error as NSError).code {

        // No agents available
        case SOSErrorCode.SOSNoAgentsAvailableError.rawValue:
            title = "Session Failed"
            description = "It looks like there are no agents available. Try again later."

        // Insufficient network error
        case SOSErrorCode.SOSInsufficientNetworkError.rawValue:
            title = "Session Failed"
            description = "Insufficient network. Check network quality and try again."

        // TO DO: Use SOSErrorCode to check for ALL other error conditions
        //         in order to give a more clear explanation of the error.
        default:
            title = "Session Error"
            description = "Unknown session error."
    }

    // Else if session stopped without an error condition...
    } else {

        switch reason {

            // Handle the agent disconnect scenario
            case .agentDisconnected:
                title = "Session Ended"
                description = "The agent has ended the session."

            // TO DO: Use SOSStopReason to check for
            //         other reasons for session ending...
            default:
                break
        }
    }


    // Do we have an error to report?
    if (title != "") {
        // TO DO: Display an alert using title & description
        NSLog("\nSOS ALERT Title: %@\nDescription: %@",title, description)
    }
}

// Delegate method for error conditions.
func sos(_ sos: SOSSessionManager!, didError error: Error!) {
    NSLog("SOS error (%d): '%@'", (error as NSError).code, error.localizedDescription)
}
}

```

Check SOS Agent Availability

Before starting a session, you can check the availability of your SOS agents and then provide your users with more accurate expectations. To use agent availability, implement the [SOSAgentAvailabilityDelegate](#) methods and start polling your org.

 **Note:** When subscribing to the agent availability delegate, it can take several seconds before any of your delegate methods are called. We don't suggest that you block a user's ability to start a session during this period.

1. Implement the [SOSAgentAvailabilityDelegate](#) methods in your `UIViewController`.

In Objective-C:

```
- (void)agentAvailability:(__weak id)agentAvailability
    didChange:(SOSAgentAvailabilityStatusType)availabilityStatus {

    // TO DO: Handle event...
}

- (void)agentAvailability:(__weak id)agentAvailability
    didError:(NSError *)error {

    // TO DO: Handle error...
}
```

In Swift:

```
func agentAvailability(_ agentAvailability: Any!,
    didChange availabilityStatus: SOSAgentAvailabilityStatusType) {
    // TO DO: Handle event...
}

func agentAvailability(_ agentAvailability: Any!,
    didError error: Error!) {
    // TO DO: Handle error...
}
```

Refer to the [SOSAgentAvailabilityStatusType](#) enumerated type for list of potential status messages.

2. Add your `UIViewController` as a delegate to the [SOSAgentAvailability](#) object and call [startPollingWithOrganizationId:deploymentId:liveAgentPod:](#).

In Objective-C:

```
SOSAgentAvailability *agentAvailability =
    [SCServiceCloud sharedInstance].sos.agentAvailability;
[agentAvailability addDelegate:self];
[agentAvailability startPollingWithOrganizationId:@"YOUR-ORG-ID"
                                     deploymentId:@"YOUR-DEPLOY-ID"
                                     liveAgentPod:@"YOUR-LA-POD"];
```

In Swift:

```
let agentAvailability = SCServiceCloud.sharedInstance().sos.agentAvailability!
agentAvailability.add(self)
agentAvailability.startPolling(withOrganizationId: "YOUR-ORG-ID",
                              deploymentId: "YOUR-DEPLOY-ID",
                              liveAgentPod: "YOUR-LA-POD")
```

This method takes the same values you specified when starting an SOS session. For more info, see [Quick Setup: SOS](#).



Example: This example accesses the Agent Availability feature and handles the appropriate events. In the code, `_sosBtn` is a `UIButton` with text that turns green when an agent is available, red when no agents are available, and gray when the status is unknown.

```
- (void)viewDidLoad {
    SOSAgentAvailability *agentAvailability =
        [SCServiceCloud sharedInstance].sos.agentAvailability;
    [agentAvailability addDelegate:self];
    [agentAvailability startPollingWithOrganizationId:@"YOUR-ORG-ID"
                                           deploymentId:@"YOUR-DEPLOY-ID"
                                           liveAgentPod:@"YOUR-LA-POD"];
}

// Delegate methods
- (void)agentAvailability:(__weak id)agentAvailability
    didChange:(SOSAgentAvailabilityStatusType)availabilityStatus {

    UIColor *color;

    switch (availabilityStatus) {
        case SOSAgentAvailabilityStatusAvailable: {
            color = [UIColor greenColor];
            [_sosBtn setEnabled:YES];
            break;
        }
        case SOSAgentAvailabilityStatusUnavailable: {
            color = [UIColor redColor];
            [_sosBtn setEnabled:NO];
            break;
        }
        case SOSAgentAvailabilityStatusUnknown:
        default: {
            color = [UIColor grayColor];
            [_sosBtn setEnabled:NO];
            break;
        }
    }
    [_sosBtn setTitleColor:color forState:UIControlStateNormal];
}

- (void)agentAvailability:(__weak id)agentAvailability
    didError:(NSError *)error {

    UIAlertView *alert = nil;
    __weak SOSLaunchViewController *weakSelf = self;

    // ensure there is no SOS session currently active
    if (error && [SOSSessionManager sharedInstance].state ==
        SOSSessionStateInactive) {

        alert = [[UIAlertView alloc] initWithTitle:@"Error"
                                                message:[error localizedDescription]
```

```
                delegate:wSelf  
                cancelButtonTitle:@"OK"  
                otherButtonTitles:nil];  
[alert show];  
}  
}
```

Enable and Disable Screen Sharing

There are some scenarios where you may want to programmatically turn off screen sharing in mid-session. You can enable and disable screen sharing using the `screenSharing` property.

You can control screen sharing using the `screenSharing` object on the `SOSSessionManager` shared instance. The `screenSharing.enabled` property enables or disables the screen sharing functionality.

The following code disables screen sharing.

In Objective-C:

```
[SCServiceCloud sharedInstance].sos.screenSharing.enabled = NO;
```

In Swift:

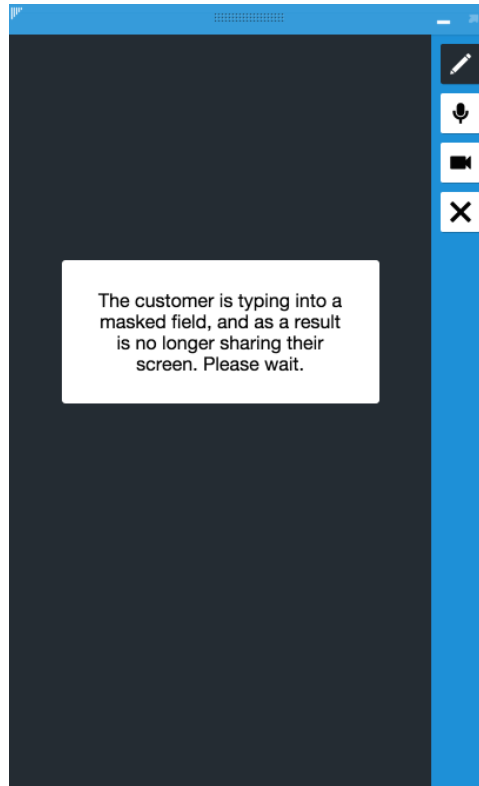
```
SCServiceCloud.sharedInstance().sos.screenSharing.enabled = false
```

Field Masking

If an application contains sensitive information that an agent shouldn't see during an SOS session, you can hide this information from the agent.

When a customer enters information into a masked field, screen sharing is disabled and the agent is notified that sharing is unavailable until the user has finished. Field masking is an integral feature to help bring your application to PII, PCI, and HIPAA compliance.

Agent's view when a field is masked:



To use field masking, replace the `UITextField` containing sensitive information with a `SOSMaskedTextField`.

When there isn't an SOS session running, a masked text field appears the same as a standard `UITextField`. However, when an SOS session is running, an `SOSMaskedTextField` appears different from a standard `UITextField` field. When the user interacts with the `SOSMaskedTextField`, screen sharing stops while the contents of that field is visible to the user. When the user finishes editing the masked field, screen-sharing resumes.

[Create Masked Field Using Storyboard](#)

To create a masked field using the storyboard, specify the `SOSMaskedTextField` custom class and set the user-defined runtime attributes.

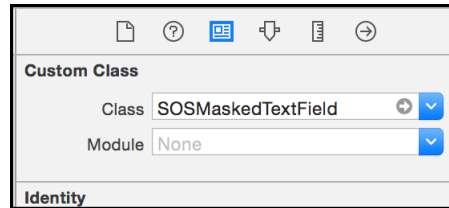
[Create Masked Field Programmatically](#)

To create a masked field manually, instantiate and style a `SOSMaskedTextField` instance.

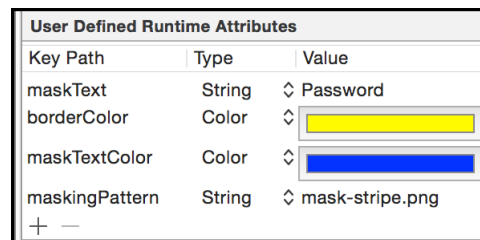
Create Masked Field Using Storyboard

To create a masked field using the storyboard, specify the `SOSMaskedTextField` custom class and set the user-defined runtime attributes.

1. Create a standard `UITextField` with the storyboard.
2. From the **Identity Inspector**, specify `SOSMaskedTextField` for the **Custom Class**.



- From the **Identity Inspector**, specify the text field style settings in the **User Defined Runtime Attributes** section.



The follow key path attributes are available:

maskPattern (String)

Name of the image file used to fill the background of the masked field when an SOS session is active

borderColor (Color)

UIColor of the border around the masked field

maskText (String)

Text to appear in the masked field

maskPattern (Color)

UIColor of the text in the masked field



Create Masked Field Programmatically

To create a masked field manually, instantiate and style a `SOSMaskedTextField` instance.

- Define an `SOSMaskedTextField` instead of a `UITextField`.

```
@interface SOSMaskedTestViewController : UIViewController

@property (strong, nonatomic) IBOutlet SOSMaskedTextField *maskCodeExample;

@end
```

- Instantiate and style the `SOSMaskedTextField`.

```
@interface SOSMaskedTestViewController () <UITextFieldDelegate>
@end

@implementation SOSMaskedTestViewController
```

```

@synthesize maskCodeExample;

- (void)viewDidLoad {
    [super viewDidLoad];

    // Set the size of the masked field and the look when masking is active
    maskCodeExample = [[SOSMaskedTextField alloc]
        initWithFrame:CGRectMake(20, 300, 200, 30)
        maskPattern:@"mask-stripeBY.png"
        borderColor:[UIColor yellowColor]
        text:@"Password"
        textColor:[UIColor blueColor]];

    // Make the field look like a UITextField created by the interface builder
    maskCodeExample.borderStyle = UITextBorderStyleRoundedRect;
    maskCodeExample.font = [UIFont systemFontOfSize:15];
    maskCodeExample.autocorrectionType = UITextAutocorrectionTypeNo;
    maskCodeExample.keyboardType = UIKeyboardTypeDefault;
    maskCodeExample.returnKeyType = UIReturnKeyDone;
    maskCodeExample.clearButtonMode = UITextFieldViewModeWhileEditing;
    maskCodeExample.contentVerticalAlignment =
        UIControlContentVerticalAlignmentCenter;

    [maskCodeExample setDelegate:self];

    [self.view addSubview:maskCodeExample];
}

```

Custom Data

Use custom data to identify customers, send error messages, issue descriptions, or identify the page the SOS session was initiated from.

When an agent receives an SOS call, it can be helpful to have information about the caller before starting the session. Use the custom data feature to identify customers, send error messages, identify the currently viewed page, or send other information. Custom data populates custom fields on the SOS Session object that is created within your Salesforce org for each SOS session initiated by a user.

Before using custom data, create the corresponding fields within the SOS Session object of your Salesforce org. To learn more, see [Create Custom Fields](#).

To use this feature, call the `setCustomFieldData` method on the `SOSOptions` object that you use to start an SOS session. The keys in this `NSMutableDictionary` should reference the API Name for fields defined in your SOS Session object and the values should reflect the desired values for those fields.




Note: If the field associated with the custom data that you specify in `setCustomFieldData` is not set up correctly in your Salesforce org, the SOS session will fail with an error.



Example: This example shows how to pass email information from your app to Service Cloud.

Before trying this example, be sure to define an "Email" custom field on the SOS Session object in your Salesforce org:

Field Information			
Field Label	Email	Object Name	SOS Session
Field Name	Email	Data Type	Email
API Name	Email__c		

 **Note:** By default, your org contains no custom field data. To learn about custom fields, see [Create Custom Fields](#).

Once you have created a custom field, you can build a dictionary object and add it to the `SOSOptions` object that you create when starting a session.

In Objective-C:

```
SOSOptions *options = [SOSOptions optionsWithLiveAgentPod:@"YOUR-POD-NAME"
                                     orgId:@"YOUR-ORG-ID"
                                     deploymentId:@"YOUR-DEPLOYMENT-ID"];

// Here we are passing the customer's email address as a String. Note the use
// of the field's API Name as the key in the map. We are only populating a single
// field here, but we may put an arbitrary number of entries into the map to
// populate multiple different fields.
NSMutableDictionary *myCustomData =
    [NSMutableDictionary dictionaryWithObjectsAndKeys:
     @"laurenboyle@example.com", @"Email__c", nil];

[options setCustomFieldData:myCustomData];
```

In Swift:

```
let options = SOSOptions(liveAgentPod: "YOUR-POD-NAME",
                        orgId: "YOUR-ORG-ID",
                        deploymentId: "YOUR-DEPLOYMENT-ID")

// Here we are passing the customer's email address as a String. Note the use
// of the field's API Name as the key in the map. We are only populating a single
// field here, but we may put an arbitrary number of entries into the map to
// populate multiple different fields.
options!.customFieldData = ["Email__c": "laurenboyle@example.com"]
```

When the user creates an SOS session, the `Email` field is prepopulated with the value specified in your custom data field.

Email	laurenboyle@example.com
-------	-------------------------

Replace the SOS UI

If you'd like to customize the SOS UI, you can create your own UI by subclassing the `UIViewController` class associated with that phase of the SOS session.

To replace the SOS UI, you'll need to register your view controllers for whichever phases you want to replace. The following phases are supported:

Onboarding (`SOSUIPhaseOnboarding`)

The onboarding process before a session starts.

Connecting (`SOSUIPhaseConnecting`)

The connecting process before a session starts.

Screen Sharing (`SOSUIPhaseScreenSharing`)

An active screen sharing session.

To register a view controller, use the `SOSOptions` object and call the `setViewControllerClass:for:` method before attempting to start a session. This method takes the class type for your view controller, and an `SOSUIPhase` enumerated type. When creating your view controller, make sure it subclasses the view controller for that phase of the session. Each view controller must also implement a protocol associated with that session phase.

Table 4: SOS Phases

Phase Name	View Controller to Subclass	Protocol to Implement
<code>SOSUIPhaseOnboarding</code>	<code>SOSOnboardingBaseViewController</code>	<code>SOSOnboardingViewController</code>
<code>SOSUIPhaseConnecting</code>	<code>SOSConnectingBaseViewController</code>	<code>SOSConnectingViewController</code>
<code>SOSUIPhaseScreenSharing</code>	<code>SOSScreenSharingBaseViewController</code> (which subclasses <code>SOSSessionBaseViewController</code>)	<code>SOSSessionViewController</code> , <code>SOSUIAgentStreamReceivable</code> , <code>SOSUILineDrawingReceivable</code>

For example, the following code overrides the onboarding UI.

In Objective-C:

```
SOSOptions *options = [SOSOptions optionsWithLiveAgentPod:@"YOUR-POD-NAME"
                                           orgId:@"YOUR-ORG-ID"
                                           deploymentId:@"YOUR-DEPLOYMENT-ID"];

// Register a custom onboarding view controller
[options setViewControllerClass:[SOSMyOnboardingViewController class]
for:SOSUIPhaseOnboarding];

// Perform other SOS configuration here
// ...

[[SCServiceCloud sharedInstance].sos startSessionWithOptions:options];
```

In Swift:

```
let options = SOSOptions(liveAgentPod: "YOUR-POD-NAME",
                        orgId: "YOUR-ORG-ID",
                        deploymentId: "YOUR-DEPLOYMENT-ID")!

// Register a custom onboarding view controller
options.setViewControllerClass(SOSOnboardingViewController.self, for: .onboarding)

// Perform other SOS configuration here
// ...

SCServiceCloud.sharedInstance().sos.startSession(with: options)
```

To learn more about configuring an SOS session, see [Configure an SOS Session](#).

For a detailed example that has a UI replacement, refer to the SOS example app: [SOS Example App](#). In particular, look at the view controller classes: <https://github.com/goinstant/ios-sdk-guides/tree/master/SOSExamples/SOS/ViewControllers>. You can turn on the custom UI in the sample app using the Custom View Controllers flags in the `SOSSettings.plist` file.

Alternatively, you can use the following code samples to get started.

SOS Onboarding Sample Code

Below you'll find some boilerplate sample code for the onboarding experience. This class must subclass `SOSOnboardingBaseViewController`.

```
- (BOOL)willHandleConnectionPrompt {
    return YES;
}

- (void)connectionPromptRequested {
    // TO DO: Show the onboarding view
}

// Call `handleStartSession:` to start a session
// Call `handleCancel:` to cancel a session

// See `SOSOnboardingBaseViewController`, `SOSOnboardingViewController`
// for additional functionality
```

SOS Connecting Sample Code

Below you'll find some boilerplate sample code for the connecting experience. This class must subclass `SOSConnectingBaseViewController`.

```
- (void)initializingNotification {
    // TO DO: Show initializing view
}

- (void)waitingForAgentNotification {
    // TO DO: Show waiting for agent view
}

- (void)agentJoinedNotification:(NSString *)name {
    // TO DO: Show agent joined notification
}

// Call `handleEndSession:` to cancel a session

// See `SOSConnectingBaseViewController`, `SOSConnectingViewController`
// for additional functionality
```

Screen Sharing Sample Code

Below you'll find some boilerplate sample code for the screen sharing experience. This class must subclass `SOSScreenSharingBaseViewController`.

```
- (BOOL)willHandleAgentStream {

    // This determines whether you wish to display an agent stream in your view.
    // If you return NO you will not receive a view containing the agent video feed.
    return YES;
}
```

```

- (BOOL)willHandleAudioLevel {
    // When this returns YES, you will receive updates about the audio level you can use
    // to implement an audio meter.
    return NO;
}

- (BOOL)willHandleLineDrawing {
    // This determines whether you want to handle line drawing during the session.
    return YES;
}

- (BOOL)willHandleRemoteMovement {
    // When this returns YES, you will receive screen space coordinates which represent
    // the center of where the agent has moved the view. You can use this to update
    // the position of your containing view.
    return NO;
}

- (void)didReceiveLineDrawView:(UIView * _Nonnull __weak)drawView {
    // TO DO: Handle line draw
}

- (void)didReceiveAgentStreamView:(UIView * _Nonnull __weak)agentStreamView {
    // TO DO: Handle stream view
}

// See `SOSSessionBaseViewController` for how to handle pause, mute, end session events

// See `SOSScreenSharingBaseViewController`, `SOSUIAgentStreamReceivable`,
//     `SOSUILineDrawingReceivable` for additional functionality

```

SDK Customizations

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

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 [SCServiceCloud](#) shared instance.

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. 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.

Notifications are another area for potential customization. With a little bit of code, you can have the SDK automatically handle Service Cloud remote notifications and display the relevant view.

Customize Colors

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

[Customize Fonts](#)

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

[Customize Images](#)

You can specify custom images used throughout the UI, along with the images used by Knowledge categories and within Knowledge articles.

[Customize Action Buttons](#)

You can determine which action buttons are visible by implementing a protocol method on `SCServiceCloudDelegate`.

[Customize and Localize Strings](#)

You can change the text 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.

[Handle Remote Notifications](#)

The SDK can handle remote notifications for events that it understands, such as case feed activity, and display the associated view.

[Launch SDK from a Web View](#)





Although this documentation mostly focuses on launching the UI from within your view controller code, you can just as easily launch the UI from a web view.







Customize Colors

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

There are several different tokens you can set:

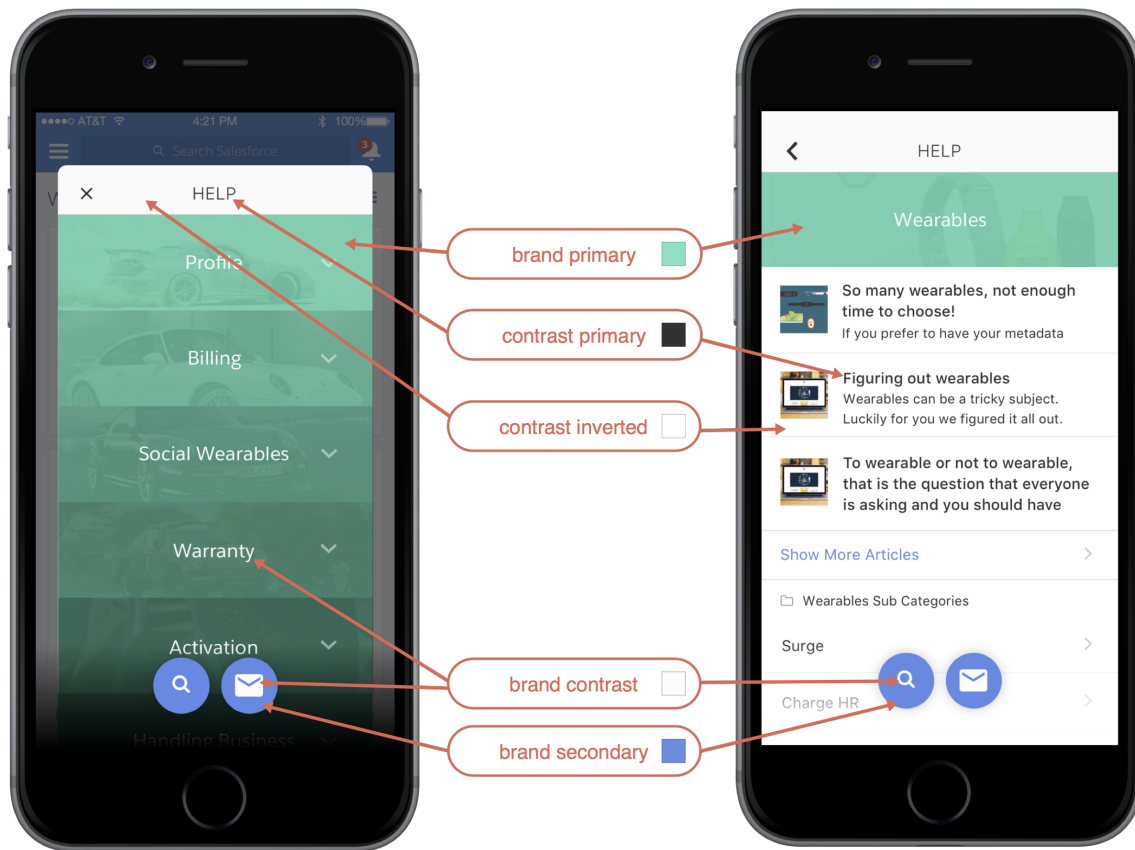
Table 5: Service SDK Colors

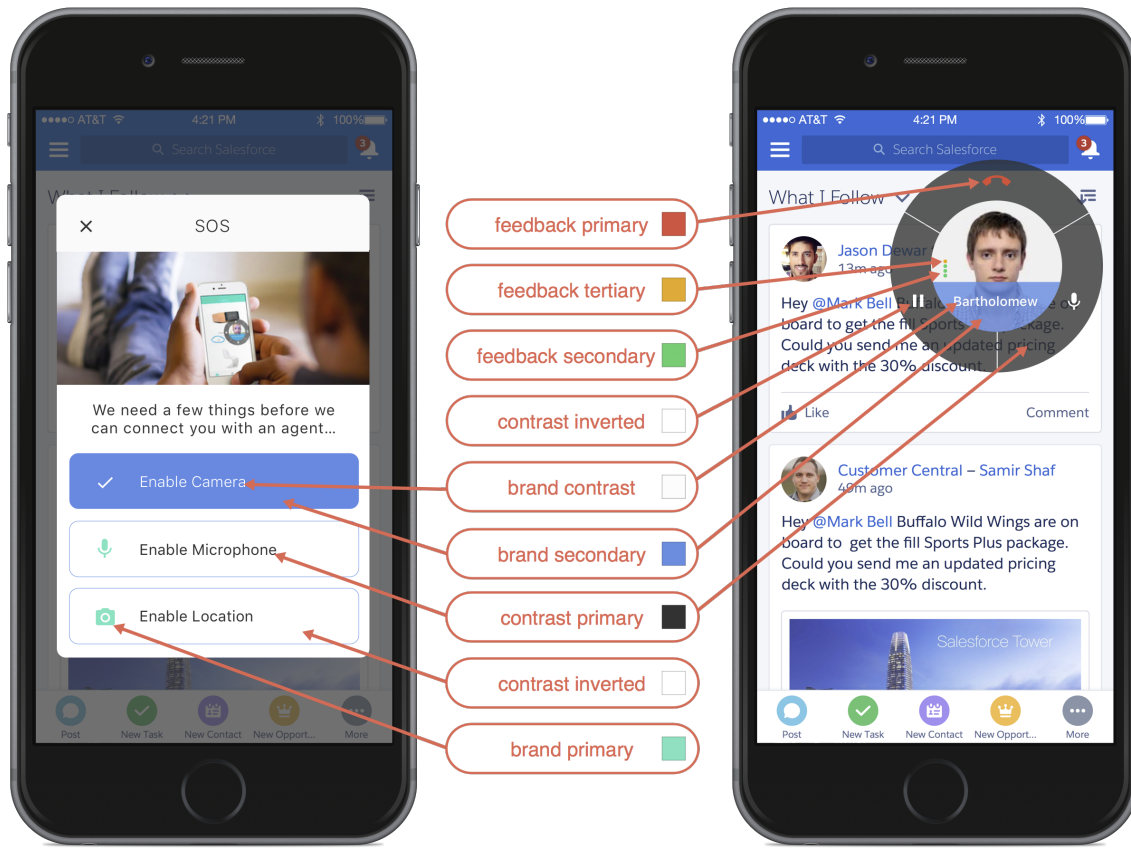
Token Name	Default Value	Sample Uses
Brand Primary <code>SCSAppearancePrimaryBrandColor</code>	#50E3C2 	In Knowledge, used for first data category, the Show More button, the footer stripe, the selected article. In SOS, used for various icons.
Brand Secondary <code>SCSAppearanceSecondaryBrandColor</code>	#4A90E2 	In Chat, used for agent text bubbles. In SOS, used as the background color for action items. Used throughout the UI for button colors.
Brand Contrast <code>SCSAppearanceBrandContrastColor</code>	#FCFCFC 	Text on areas where a brand color is used for the background. In SOS, used as the color of the icons (when they are selected).
Contrast Primary <code>SCSAppearanceContrastPrimaryColor</code>	#333333 	Primary body text color. Background color for buttons on the SOS UI.

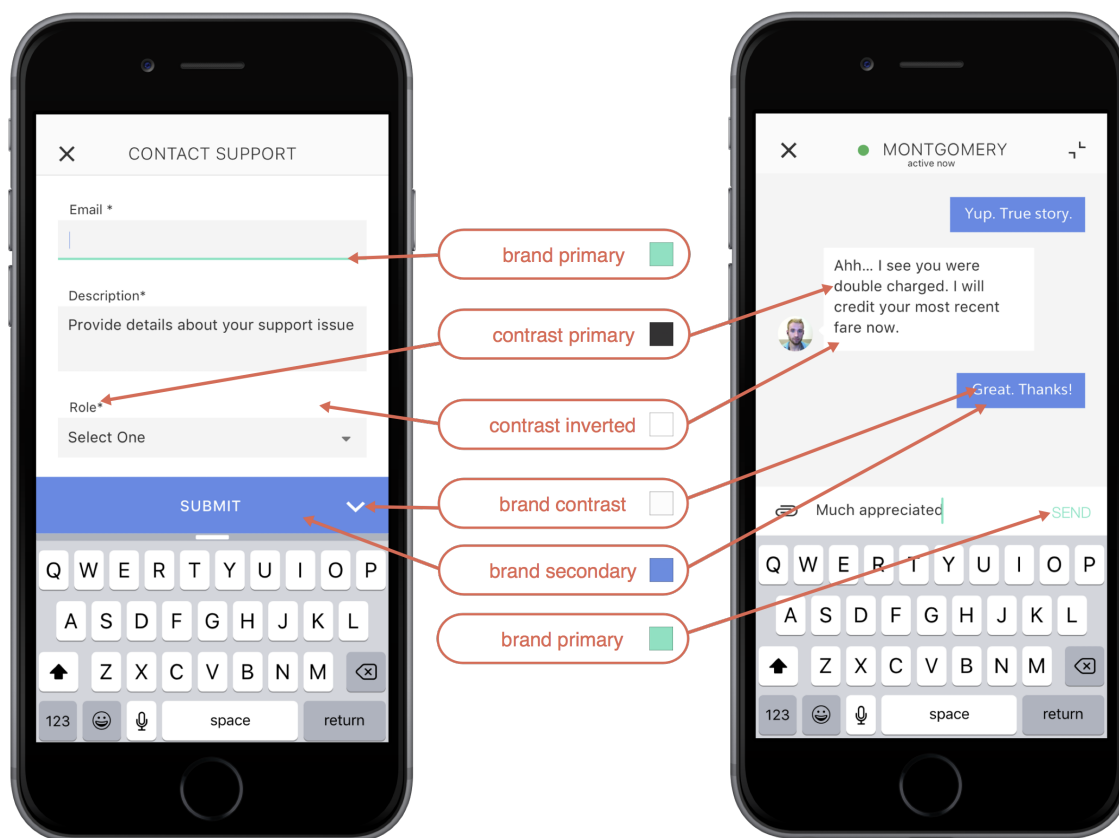
Token Name	Default Value	Sample Uses
Contrast Secondary <code>SCSAppearanceContrastSecondaryColor</code>	#767676 	Used for subcategory headers in Knowledge.
Contrast Inverted <code>SCSAppearanceContrastInvertedColor</code>	#FFFFFF 	Used for page background, navigation bar, table cell background. In SOS, used as the color of the icons (when they aren't selected).
Feedback Primary <code>SCSAppearanceFeedbackPrimaryColor</code>	#E74C3C 	Text color for error messages. In SOS, used as the mute indicator and the disconnect icon.
Feedback Secondary <code>SCSAppearanceFeedbackSecondaryColor</code>	#2ECC71 	SOS connection quality indicators, background color for the Resume button when the two-way camera is active.
Feedback Tertiary <code>SCSAppearanceFeedbackTertiaryColor</code>	#F5A623 	SOS connection quality indicators.
Overlay <code>SCSAppearanceOverlayColor</code>	<code>SCSAppearanceContrastPrimaryColor</code> (at 50% alpha)	Used for background for the Knowledge home screen.
Title Text <code>SCSAppearanceTitleTextColor</code>	#FBFBFB 	In Knowledge, used for text on data category headers and the chevron on the Knowledge home page.
Navigation Bar Background <code>SCSAppearanceNavbarBackgroundColor</code>	#FAFAFA 	Background color for the navigation bar.

Two other colors are created dynamically (`SCSAppearanceContrastTertiaryColor`, `SCSAppearanceContrastQuaternaryColor`) as a combination of the contrast inverted color (`SCSAppearanceContrastInvertedColor`) and the secondary contrast color (`SCSAppearanceContrastSecondaryColor`). These colors are used for various background settings.

These screenshots illustrate some of the ways that color branding is used throughout the user interface.







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 `SCServiceCloud` shared instance.

In Objective-C:

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

// Customize color tokens
[config setColor:UIColor.VALUE forName:TOKEN_NAME];

// Add other customizations here...

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

In Swift:

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

// Customize color tokens
config.setColor(UIColor.VALUE, forName: TOKEN_NAME)

// Add other customizations here...
```



```
// Save configuration instance
SCServiceCloud.sharedInstance().appearanceConfiguration = config
```



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

In Objective-C:

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

// Customize color tokens
[config setColor:[UIColor colorWithRed:80/255 green:227/255 blue:194/255 alpha:1.0]
    forName:SCSAppearancePrimaryBrandColor];
[config setColor:[UIColor colorWithRed:74/255 green:144/255 blue:226/255 alpha:1.0]
    forName:SCSAppearanceSecondaryBrandColor];
[config setColor:[UIColor colorWithRed:252/255 green:252/255 blue:252/255 alpha:1.0]
    forName:SCSAppearanceBrandContrastColor];

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

In Swift:

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

// Customize color tokens
config.setColor(UIColor(red: 80/255, green: 227/255, blue: 194/255, alpha: 1.0),
    forName: SCSAppearancePrimaryBrandColor)
config.setColor(UIColor(red: 74/255, green: 144/255, blue: 226/255, alpha: 1.0),
    forName: SCSAppearanceSecondaryBrandColor)
config.setColor(UIColor(red: 252/255, green: 252/255, blue: 252/255, alpha: 1.0),
    forName: SCSAppearanceBrandContrastColor)

// Save configuration instance
SCServiceCloud.sharedInstance().appearanceConfiguration = config
```

Customize Fonts

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

You can customize three font settings used throughout the Service 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, Live Agent Chat text, Knowledge data category cell in detail view,

Font Setting	Default Value	Samples Uses in the SDK
		Knowledge "show more" article footer, Knowledge "show more" button cell
SCFontWeightBold	Helvetica Neue - Semibold	Knowledge category headers, Knowledge article cell title, Case Management field labels, Case Management submit button, title of error messages

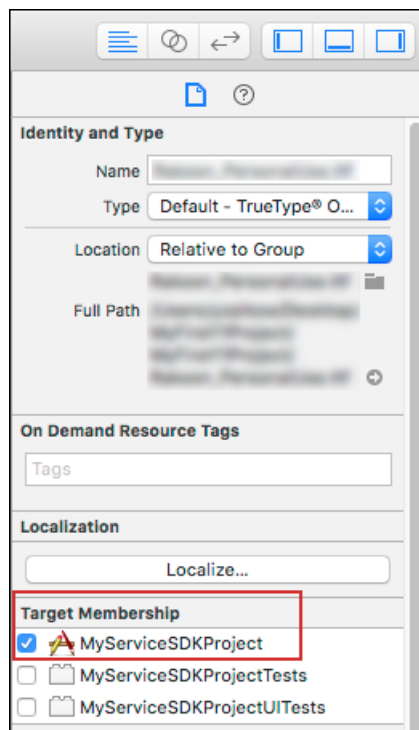
To configure your app to use different fonts:

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 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 `SCServiceCloud` shared instance.

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;
```

Swift Example:

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

// Customize font
let descriptor = UIFontDescriptor(fontAttributes:
    [UIFontDescriptorFamilyAttribute : "Proxima Nova"])
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

You can specify custom images used throughout the UI, along with the images used by Knowledge categories and within Knowledge articles.

The method to customize an image is different depending on the type of image you want to customize. This table describes what customization techniques are supported for each image type.

Table 6: Custom Image Types

Image Type	How to Customize
Stock images	Use the setImage method on the SCAppearanceConfiguration object to replace a stock image with your image. Use the named constant value for the image you intend to replace. See Replacing Stock Images for more guidance.
Data category image (Knowledge)	Data category images and article images can be customized in either one of two different ways: 1. Put the images in a specific image folder . This technique uses the imageFolderPath property on the SCSServiceConfiguration object. For data category images, the image name must match the unique name for that category. For article images, the image name must match the article number. See Supplying Knowledge Images using an Image Folder for more guidance. 2. Supply images by implementing a delegate method. This technique uses the SCKnowledgeInterfaceDelegate protocol. See Supplying Knowledge Images with a Delegate for more guidance.
Article image (Knowledge)	

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

Replacing Stock Images

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

Table 7: Stock Image Constant Values

Image Description	Named Constant Value
ServiceCore Images	<i>Images used throughout the SDK</i>
Close button	Close
Done button	Done
Error	Error
No connection	NoConnection
Minimize button (Knowledge and Live Agent Chat)	MinimizeButtonImage
Next field button (Case Publisher and Live Agent Chat)	SubmitButtonNext

Image Description	Named Constant Value
Previous field button (Case Publisher and Live Agent Chat)	SubmitButtonPrevious
Knowledge Images	<i>Images used by Knowledge</i>
Article is empty	EmptyArticle
Category header arrow	CategoryHeaderArrow
Placeholder image before search completes	SearchPlaceholder
Search action button	ActionSearch
Case Management Images	<i>Images used by Case Management</i>
Case publisher success message	CaseSubmitSuccess
Case publisher action button	ActionCasePublisher
Pick case list button	PicklistDropdown
Live Agent Chat Images	<i>Images used by Live Agent Chat</i>
Attachment button when the user can attach a file	AttachmentClipIcon
Avatar used for the agent	ChatAgentAvatar
Chat icon used in minimized view	ChatIcon
Icon used in the pre-chat screen	PreChatIcon
SOS Images	<i>Images used by SOS</i>
Cancel icon	cancel
Confirm icon	confirm
End icon	end
Pause icon	pause
Resume icon	resume
Microphone icon	mic
User muted icon	muted
Agent muted icon	agentMuted
Camera icon	camera
Agent placeholder icon (must be 61x55 pixels @1x, 122x110 @2x, 183x165 @3x)	avatar
Expand icon	downCaret
Flashlight icon	flashlight
Info icon	tip

Image Description	Named Constant Value
Mask stripe	mask-stripe

In Objective-C:

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

// Specify images
[config setImage:MY_CUSTOM_IMAGE compatibleWithTraitCollection: MY_TRAITS
    forName: NAMED_CONSTANT_VALUE];

// Add other customizations here...

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

In Swift:

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

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

// Add other customizations here...

// Save configuration instance
SCServiceCloud.sharedInstance().appearanceConfiguration = config
```

Supplying Knowledge Images using an Image Folder

For knowledge articles images and data category images, specify the location of the image using the [imageFolderPath](#) property on the [SCSServiceConfiguration](#) object. We suggest you do this at the time that you configure the object to connect to your org. To learn more about connecting to your org, see [Quick Setup: Knowledge](#).

For data category images, the image name must match the unique name for that category. For article images, the image name must match the article number.

Here is some sample code to get you started.

In Objective-C:

```
// Create configuration object with init params
SCSServiceConfiguration *config = [[SCSServiceConfiguration alloc]
    initWithCommunity:[NSURL URLWithString:@"https://mycommunity.example.com"]
    dataCategoryGroup:@"Regions"
    rootDataCategory:@"All"];

// Specify image folder
config.imageFolderPath = @"my/path/";
```

```
// Pass configuration to shared instance
[SCServiceCloud sharedInstance].serviceConfiguration = config;
```

In Swift:

```
// Create configuration object with init params
let config = SCServiceConfiguration(
    community: URL(string: "https://mycommunity.example.com")!,
    dataCategoryGroup: "Regions",
    rootDataCategory: "All")

// Specify image folder
config.imageFolderPath = "my/path/"

// Pass configuration to shared instance
SCServiceCloud.sharedInstance().serviceConfiguration = config
```

Supplying Knowledge Images with a Delegate

For knowledge articles images and data category images, you can instead supply images using a delegate.

1. Provide the `SCServiceCloud` instance with an implementation of `SCKnowledgeInterfaceDelegate`.

In Objective-C:

```
[SCServiceCloud sharedInstance].knowledge.delegate =
    mySCKnowledgeInterfaceDelegateImplementation;
```

In Swift:

```
SCServiceCloud.sharedInstance().knowledge.delegate =
    mySCKnowledgeInterfaceDelegateImplementation
```

2. Implement the two delegate methods.

In Objective-C:

```
- (UIImage*)knowledgeInterface:(SCKnowledgeInterface *)interface
    imageForArticle:(NSString *)articleId
compatibleWithTraitCollection:(UITraitCollection *)traitCollection {

    // Your code that returns an image given an article id

    return myImage;
}

- (UIImage*)knowledgeInterface:(SCKnowledgeInterface *)interface
    imageForDataCategory:(NSString *)categoryName
compatibleWithTraitCollection:(UITraitCollection *)traitCollection {

    // Your code that returns an image given a data category

    return myImage;
}
```

In Swift:

```
func knowledgeInterface(_ interface: SCKnowledgeInterface,
    imageForArticle articleId: String,
    compatibleWith traitCollection: UITraitCollection?)
    -> UIImage? {

    // Your code that returns an image given an article id

    return myImage
}

func knowledgeInterface(_ interface: SCKnowledgeInterface,
    imageForDataCategory categoryName: String,
    compatibleWith traitCollection: UITraitCollection?)
    -> UIImage? {

    // Your code that returns an image given a data category

    return myImage
}
```

Customize Action Buttons

You can determine which action buttons are visible by implementing a protocol method on `SCServiceCloudDelegate`.



Note: This customization can be used for Knowledge and Case Management.

When dealing with action buttons, use the named constant value for the button you'd like to show or hide:

SCSActionCasePublisher

Allows a user to create a case. If the user is authenticated, this will show a list of their existing cases.

SCSActionArticleSearch

Allows a user to perform a search for a knowledge base article.

To control visibility:

1. Implement the [SCServiceCloudDelegate](#) protocol and handle the [shouldShowActionWithName](#) method. This method is called to determine whether the specified action button should be present, when applicable.

In Objective-C:

```
- (BOOL)serviceCloud:(SCServiceCloud *)serviceCloud
    shouldShowActionWithName:(NSString *)name {

    BOOL shouldShowAction = NO;

    if ([name isEqualToString:SCSActionCasePublisher]) {
        // Determine whether we should show the action for case publisher
        // If so, set shouldShowAction to YES; otherwise NO.
        shouldShowAction = YES;
    } else if ([name isEqualToString:SCSActionArticleSearch]) {
        // Determine whether we should show the action for article searches
        // If so, set shouldShowAction to YES; otherwise NO.
        shouldShowAction = YES;
    }
}
```



```

    }

    return shouldShowAction;
}

```

In Swift:

```

func serviceCloud(_ serviceCloud: SCServiceCloud,
    shouldShowActionWithName name: String) -> Bool {

    var shouldShowAction = false

    switch (name) {
        case SCSCActionCasePublisher:
            // Determine whether we should show the action for case publisher...
            // If so, set shouldShowAction to true; otherwise false.
            shouldShowAction = true
        case SCSCActionArticleSearch:
            // Determine whether we should show the action for article searches...
            // If so, set shouldShowAction to true
            shouldShowAction = true
        default:
            shouldShowAction = false
    }

    return shouldShowAction
}

```

Be sure to enable the associated feature (knowledge, cases) before attempting to show a button for that feature. Otherwise, the button will not be visible. For example, before a case-related button can appear, you need to enable the feature first:

```
[SCServiceCloud sharedInstance].cases.enabled = YES;
```

Customize and Localize Strings

You can change the text 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.

Service 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:

ServiceCore String Constants

List of string constants that are shared across all Service SDK frameworks.

KnowledgeFramework String Constants

List of string constants associated with the Service SDK Knowledge framework.

CasesFramework String Constants

List of string constants associated with the Service SDK Cases framework.

ChatFramework String Constants

List of string constants associated with the Service SDK Live Agent Chat framework.

SOSFramework String Constants

List of string constants associated with the Service SDK SOS framework.

The following languages are currently supported:

Table 8: Supported Languages

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

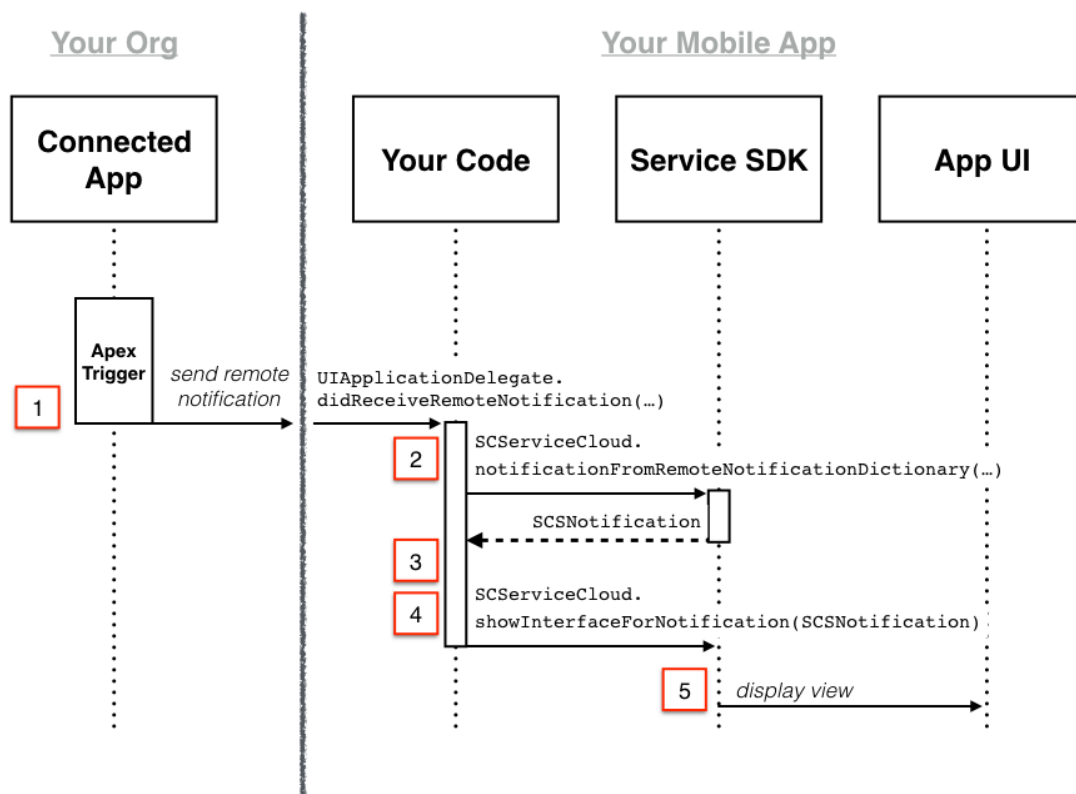
Language Code	Language
zh	Chinese

Handle Remote Notifications

The SDK can handle remote notifications for events that it understands, such as case feed activity, and display the associated view.

When building your app, you may have remote notifications arrive from many different sources. Some of these sources may be associated with your Salesforce org. You can ask the SDK whether it can handle a notification, and if it can, you can tell the SDK to show the appropriate view.

The sequence diagram below illustrates a scenario where an Apex trigger from your connected app sends a notification to your mobile app and you instruct the SDK to display the associated view.



To deal with push notifications from within your app:

1. Send the push notification from your org.

For help sending push notifications for case feed activity, see [Push Notifications for Case Activity](#).

- When you receive a remote notification (from your app delegate's `didReceiveRemoteNotification` method), pass notification information to `notificationFromRemoteNotificationDictionary:` to determine whether the SDK can handle the notification.

In Objective-C:

```
- (void)application:(UIApplication *)application
    didReceiveRemoteNotification:(NSDictionary *)userInfo
    fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler {

    SCSNotification *notification =
        [[SCSServiceCloud sharedInstance]
         notificationFromRemoteNotificationDictionary:userInfo];

    // TO DO: Handle notification here
}
```

In Swift:

```
func application(_ application: UIApplication,
    didReceiveRemoteNotification userInfo: [AnyHashable : Any],
    fetchCompletionHandler completionHandler: @escaping
    (UIBackgroundFetchResult)
    -> Void) {

    let notification =
        SCSServiceCloud.sharedInstance().notification(fromRemoteNotificationDictionary:
        userInfo)

    // TO DO: Handle notification here
}
```

This method returns `nil` if the SDK can't handle the notification; it returns an `SCSNotification` object if it *can* handle the notification.

- (Optional) If you want to handle the notification by yourself (like to display your own view), you can inspect the notification type to determine what feature area the notification is associated with. This property returns a `SCSNotificationType` enum.

```
notification.notificationType
```

When dealing with case activity notifications, this value is `SCSNotificationTypeCase`.

- If you want the SDK to handle the notification for you, call `showInterfaceForNotification:` and pass in the `SCSNotification` object.

In Objective-C:

```
[[SCSServiceCloud sharedInstance] showInterfaceForNotification:notification];
```

In Swift:

```
SCSServiceCloud.sharedInstance().showInterface(for: notification)
```

- The SDK displays the relevant view for the notification.

When dealing with case activity notifications, this command displays the Case List or Case Details screen (depending on the notification), showing the relevant case information. If the appropriate screen is already showing, it refreshes.

Launch SDK from a Web View

Although this documentation mostly focuses on launching the UI from within your view controller code, you can just as easily launch the UI from a web view.

These instructions assume that you are already familiar with how to launch the UI for the feature you are using:

- [Quick Setup: Knowledge](#)
- [Quick Setup: Case Publisher as a Guest User](#)
- [Quick Setup: Live Agent Chat](#)
- [Quick Setup: SOS](#)

Once you're familiar with the feature, use this documentation to launch the Service SDK UI from a web view.

1. Create a view controller with a single `UIWebView`.
2. Have your view controller implement the `UIWebViewDelegate` protocol.

In Objective-C:

```
@interface ViewController : UIViewController <UIWebViewDelegate>
```

In Swift:

```
class ViewController: UIViewController, UIWebViewDelegate {
```

3. Set your view controller as the web view delegate and configure your Service SDK feature.

In Objective-C:

```
- (void)viewDidLoad {
    [super viewDidLoad];

    // Point the web view to your web application.
    NSURL *url = [NSURL URLWithString:@"https://my.web.app"];
    NSURLRequest *request = [NSURLRequest requestWithURL:url];
    [webView loadRequest:request];

    // Make sure you set your view controller as a delegate to
    // your webview so you can trap requests.
    [webView setDelegate:self];

    // TO DO: Set up and configure your Service SDK feature...
    // Use SCSServiceConfiguration or SCSChatConfiguration or SOSOptions
    // to configure your feature and point it to your org.
    // See the "Quick Setup" instructions for the appropriate feature.
}
```

In Swift:

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Point the web view to your web application.
    let url = URL(string: "https://my.web.app")
    let request = URLRequest(url: url!)
    webView.loadRequest(request)
```

```

// Make sure you set your view controller as a delegate to
// your webView so you can trap requests.
webView.delegate = self

// TO DO: Set up and configure your Service SDK feature...
// Use SCSServiceConfiguration or SCSSChatConfiguration or SOSOptions
// to configure your feature and point it to your org.
// See the "Quick Setup" instructions for the appropriate feature.
}

```

4. Add a delegate handler for parsing requests.

Implement the `webView:shouldStartLoadWithRequest:navigationType:` method to trap requests.

In Objective-C:

```

- (BOOL)webView:(UIWebView *)webView
    shouldStartLoadWithRequest:(NSURLRequest *)request
        navigationType:(UIWebViewNavigationType)navigationType {

    NSURL *url = [request URL];

    // For this example, we use "servicesdk" as the URL scheme...
    if ([[url scheme] isEqualToString:@"servicesdk"]) {

        // "servicesdk://start", corresponds to the host...
        if ([[url host] isEqualToString:@"start"]) {

            // TO DO: Launch the SDK here...
            // Use setInterfaceVisible or startSessionWithOptions
            // or startSessionWithConfiguration, depending on what you're launching.
            // See the "Quick Setup" instructions for the appropriate feature.
        }

        // Returning NO here ensures that the browser doesn't
        // try to do anything with this request.
        return NO;
    }

    return YES;
}

```

In Swift:

```

func webView(_ webView: UIWebView,
    shouldStartLoadWith request: URLRequest,
    navigationType: UIWebViewNavigationType) -> Bool {

    let url = request.url

    // For this example, we use "servicesdk" as the URL scheme...
    if (url?.scheme == "servicesdk") {

        // "servicesdk://start", corresponds to the host...
        if (url?.host == "start") {

```

```

        // TO DO: Launch the SDK here...
        // Use setInterfaceVisible or startSessionWithOptions
        // or startSessionWithConfiguration, depending on what you're launching.
        // See the "Quick Setup" instructions for the appropriate feature.
    }

    // Returning false here ensures that the browser doesn't
    // try to do anything with this request.
    return false
}

return true
}

```

5. (Optional) If you want to send additional context to your session (such as an email address), you can add a query parameter to the URL (for example, `servicesdk://start?email=new@example.com`), and then parse the parameter as you can see in this code sample.

In Objective-C:

```

- (BOOL)webView:(UIWebView *)webView
    shouldStartLoadWithRequest:(NSURLRequest *)request
        navigationType:(UIWebViewNavigationType)navigationType {

    NSURL *url = [request URL];
    if ([[url scheme] isEqualToString:@"servicesdk"]) {

        // Here's the new code to parse a basic query containing an email.
        if ([url query]) {

            // Very simple example that only handles one parameter.
            // In this example query would look like 'email=new@example.com'.
            NSString *query = [url query];

            // Split on = and get the second component.
            NSString *email = [query componentsSeparatedByString:@"="][1];

            // In general, you'll want to make sure that this is decoded properly.
            email = [email
                stringByReplacingPercentEscapesUsingEncoding:NSUTF8StringEncoding];

            // TO DO: Do something with this information.
        }

        if ([[url host] isEqualToString:@"start"]) {

            // TO DO: Launch the SDK here...
            // Use setInterfaceVisible or startSessionWithOptions
            // or startSessionWithConfiguration, depending on what you're launching.
            // See the "Quick Setup" instructions for the appropriate feature.
        }

        // Returning NO here ensures that the browser doesn't
        // try to do anything with this request.
        return NO;
    }
}

```

```

    }

    return YES;
}

```

In Swift:

```

func webView(_ webView: UIWebView,
             shouldStartLoadWith request: URLRequest,
             navigationType: UIWebViewNavigationType) -> Bool {

    let url = request.url

    if (url?.scheme == "servicesdk") {

        // Here's the new code to parse a basic query containing an email.
        if (url?.query != nil) {

            // Very simple example that only handles one parameter.
            // In this example query would look like 'email=new@example.com'.
            let query = url?.query!

            // Split on = and get the second component.
            var email = query?.components(separatedBy: "=")[1]

            // In general, you'll want to make sure that this is decoded properly.
            email = email?.removingPercentEncoding

            // TO DO: Do something with this information.
        }

        if (url?.host == "start") {

            // TO DO: Launch the SDK here...
            // Use setInterfaceVisible or startSessionWithOptions
            // or startSessionWithConfiguration, depending on what you're launching.
            // See the "Quick Setup" instructions for the appropriate feature.
        }

        // Returning false here ensures that the browser doesn't
        // try to do anything with this request.
        return false
    }

    return true
}

```

6. Add the code to your web page that calls the URL. You can do this using HTML or JavaScript.
 - a. Call the URL using an HTML anchor tag.

```
<a href="servicesdk://start">Get help!</a>
```


If you want to include additional information, add it to the `href`.

```
<a href="servicesdk://start?email=new@example.com">Get help!</a>
```

- b. Call the URL using a JavaScript function.

```
function myFunc() {  
    window.location = "servicesdk://start"  
}
```

Troubleshooting

Get some guidance when you run into issues.

[Logging](#)

Learn how to turn on and control the logging framework so that you can debug issues during development.

[Unable to Access My Community](#)

What to do when you can't seem to get to your community from within your app.

[My App Was Rejected](#)

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

[SOS Network Troubleshooting Guide](#)

If you can't connect with an SOS agent from your app, you have network connectivity issues, possibly related to your firewall or proxy.

Logging

Learn how to turn on and control the logging framework so that you can debug issues during development.

If you want to inspect the Service SDK logs, you can turn on logging using the `setLogLevel:forIdentifiersWithPrefix:` method in the `SFLogger` shared instance.

In Objective-C:

```
SFLogger *logger = [SFLogger sharedLogger];  
[[SFLogger sharedLogger] setLogLevel:SFLogLevelWarning  
    forIdentifiersWithPrefix:@"com.salesforce.ServicesDK"];
```

In Swift

```
let logger = SFLogger.shared()  
logger.setLogLevel(SFLogLevel.warning,  
    forIdentifiersWithPrefix:"com.salesforce.ServiceSDK")
```

Log Level

The log level is specified in the `setLogLevel:forIdentifiersWithPrefix:` method using the `SFLogLevel` enumerated type. It can be one of these values:

- `SFLogLevelVerbose`

- `SFLogLevelDebug`
- `SFLogLevelInfo`
- `SFLogLevelWarning`
- `SFLogLevelError`
- `SFLogLevelOff`

By default, the level is set to `SFLogLevelError` so that only serious issues are logged.

Log Output

By default, logs only go to the Xcode output. You can have logs go to ASL (Apple System Logger) using the `logToASL` property. You can use the `logToFile` property to log messages to a file in your app's directory on the device. This feature supports log rotation every 48 hours, with a maximum of three log files on a device. If you turn off `logToFile`, the files are automatically removed.

In Objective-C:

```
logger.logToFile = YES;
```

In Swift:

```
logger.shouldLogToFile = true
```

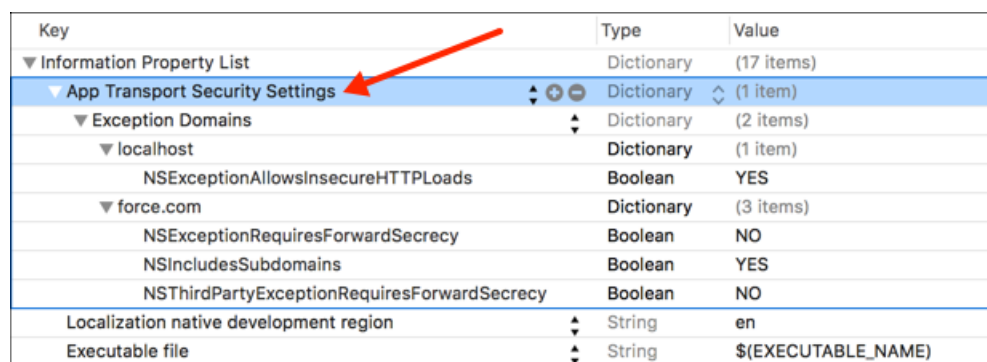
You can programmatically access this file with the `logFileContents` property.

Unable to Access My Community

What to do when you can't seem to get to your community from within your app.

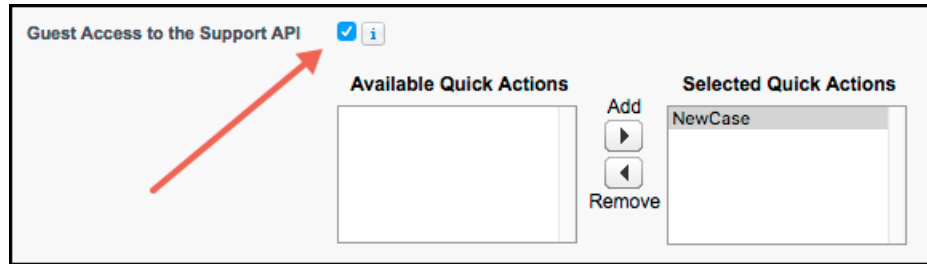
Run through this checklist to help diagnose the root cause.

1. Does your `SCSServiceConfiguration` object point to a valid, accessible community URL?
2. Have you set up **App Transport Security** (ATS) exceptions for your community's domain and for `localhost`? See [Install the SDK](#) for more info.



Key	Type	Value
▼ Information Property List	Dictionary	(17 items)
▼ App Transport Security Settings	Dictionary	(1 item)
▼ Exception Domains	Dictionary	(2 items)
▼ localhost	Dictionary	(1 item)
NSExceptionAllowsInsecureHTTPLoads	Boolean	YES
▼ force.com	Dictionary	(3 items)
NSExceptionRequiresForwardSecrecy	Boolean	NO
NSIncludesSubdomains	Boolean	YES
NSThirdPartyExceptionRequiresForwardSecrecy	Boolean	NO
Localization native development region	String	en
Executable file	String	\$(EXECUTABLE_NAME)

3. Have you set up a Community or Force.com site? See [Cloud Setup for Knowledge](#) for more info.
4. Do you have "Guest Access to the Support API" enabled for your site? See [Cloud Setup for Knowledge](#) for more info.



5. (For Knowledge only) Do you have **Knowledge** enabled in your org? Do you have Knowledge licenses? See [Cloud Setup for Knowledge](#) for more info.
6. (For Knowledge only) Is the user setting up the knowledge base enabled as a **Knowledge User**? See [Cloud Setup for Knowledge](#) for more info.

7. (For Knowledge only) Have you made the article types, the data categories, and the article layout fields visible to guest users? See [Guest User Access for Your Community](#) for more info.

Object Permissions	
Permission Name	Enabled
Read	<input checked="" type="checkbox"/>
Create	<input checked="" type="checkbox"/>
Edit	<input checked="" type="checkbox"/>
Delete	<input checked="" type="checkbox"/>

Field Permissions		
Field Name	Read Access	Edit Access
Additional Information	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Additional Resources	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Archived By	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Archived Date	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

8. (For Knowledge only) Have you made your articles accessible to the **Public Knowledge Base** channel? See [Cloud Setup for Knowledge](#) for more info.

Article Assignment	
Assigned To	<input type="text"/>
Assigned By	<input type="text"/>
Instructions	--
Assignment Due Date	--

Article Properties	
Publishing Status	Draft
Type	<input type="text"/>
Article Number	000001001
Created By	<input type="text"/>
Last Modified By	<input type="text"/> 6/1/2016 2:49 PM

Categories	
CategoryGroup	<input type="text"/> <input type="button" value="Edit"/>

Channels	
<input checked="" type="checkbox"/>	Internal App
<input type="checkbox"/>	Partner
<input type="checkbox"/>	Customer
<input checked="" type="checkbox"/>	Public Knowledge Base

My App Was Rejected

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

- If you receive errors related to unsupported architectures when you upload your app to the App Store, it may be because you didn't strip unneeded architectures from the dynamic libraries used by the Service SDK. See [Prepare Your App for Submission](#) for more information.
- 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.

SOS Network Troubleshooting Guide

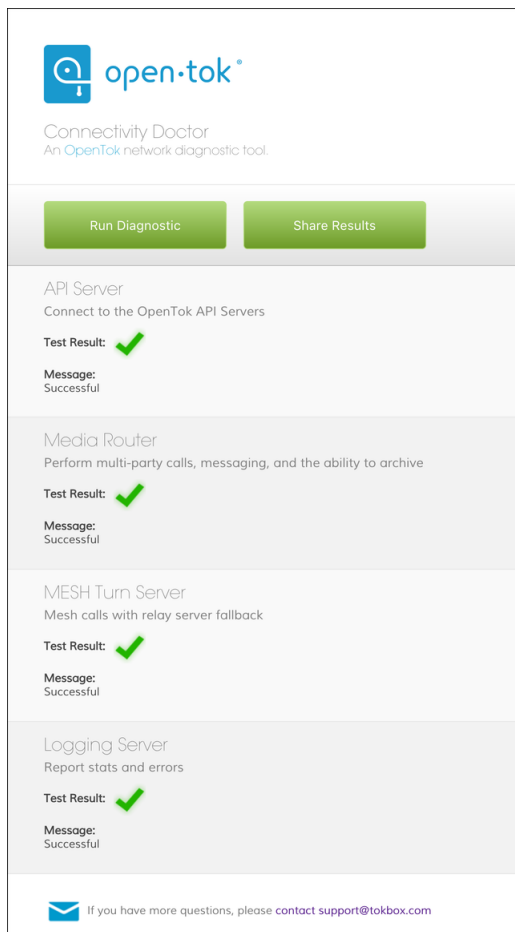
If you can't connect with an SOS agent from your app, you have network connectivity issues, possibly related to your firewall or proxy.

SOS uses the [Tokbox](#) OpenTok platform to provide screen sharing and video communication during an SOS session. These guidelines can help you diagnose whether the problem is linked to a networking issue and how to send us diagnostic information if necessary.

Step 1: Run Connectivity Doctor

The Tokbox [Connectivity Doctor](#) tests for connectivity issues. You can access this tool via the [web](#), an [iOS app](#), or an [Android app](#). This tool tests network issues in these areas.

1. API server – Session initialization and signaling tests
2. Media router – Whether you can access Tokbox media servers
3. MESH turn server – Relay server fallback mechanism
4. Logging server – Communication of stats and errors to the Tokbox logging server



If all tests pass, go to step 2. If any test fails, you probably need to configure your ports.

API Server or Logging Server Issues

OpenTok clients use HTTP and WSS connections from the client browser to the OpenTok servers on port **TCP/443**. If the only way to access the internet from your network is through a proxy, it must be a transparent proxy. Make sure that TCP/443 is open.

Media Router or Mesh Turn Issues

OpenTok clients can use UDP or TCP connections for media. Salesforce recommends that UDP is enabled to improve the quality of real-time audio and video communications. This connection is bidirectional but always initiated from the client so an external entity can't send malicious traffic in the opposite direction.

- Best experience: We recommend that you open **UDP ports 1025 - 65535**.
- Good experience: Open **UDP port 3478**.
- Minimum experience: Open **TCP port 443**. Some firewall or proxy rules only allow for SSL traffic over port 443. Make sure that non-web traffic can also pass over this port.

Step 2: Test the Tokbox Chat Room

Tokbox has a [public-facing chat room site \(https://opentokrtc.com/\)](https://opentokrtc.com/) that you can use for normal chatting. You can also use this site as a test tool.

1. From the chat room site, create a room called "example". Join that room or click [here](#). You should then see yourself on video. If the video isn't present, make sure that you've given access to the camera and microphone. If nothing happens, go to step 3.

2. Open another browser tab and enter the same URL link as in the previous browser tab (for example, <https://opentokrtc.com/room/example>). You should see a two-way audio and video-enabled chat. If this process doesn't work, go to step 3.

If you can have a two-way chat, your ports are configured properly and you're done.

Step 3: Gather JSON Metadata

In this step, we'll gather metadata about the failed chat room session. To get this information, open a browser tab and enter <https://opentokrtc.com/example.json>. If you created a room with a different name, replace the word "example" in the URL accordingly. You'll see JSON content similar to this example.

```
{ "apiKey": "45599822", "token": "T1==cGyydG51cl9pZD00NTU5OTgyMiZzaWc9NTcyOWI4NjJhOT
diN2EwYWRmMjZkZjI5MzkxZjkwMjdlNmM0ODNiMTpZzXNzaW9uX2lkPTFfTVg0ME5UVTVPVGd5TW41LU
1UUTNPVEUyTWpVeK1USTVOBjQ0WkcxSFp6VnNVMnBMZFRKVFp6SmhaRlZ6WVhvMldVTi1mZyZjcmVhdG
VfdGltZT0xNDc5MTY0MzA4Jm5vbmNlPTAuMzgZODc1MjEwMzAzODEzMiZyb2x1PXB1Ymxpc2hlciZleH
BpcmVfdGltZT0xNDc5MjUwNzA4Jm5vbmNlY3Rpb25fZGF0YT01N0I1MjJlc2VyTmFtZSUyMiUzQSUyMk
Fub255bW91cyUyMFVzZXI3MiUyMiU3RA==", "username": "Anonymous User72", "firebaseURL":
"https://ot-archiving.firebaseio.com/sessions//1_MX40NTU5OTgyMn5-MTQ3OTE2MjUzMTI5Nn43ZG1HZzVsU2pLdTJTZzJhZGVzYXo3WUN-fg", "firebaseToken": "eyJ0eXAiOiJKV1QiLCJhbG
ciOiJIUzI1NiJ9.eyJ2IjowLCJkIjpw7InVpZCI6IkFub255bW91cyBVc2VyNzIwLjE3NDUwOTM5OTg3N
zQ4ODYiLCJzZXNzaW9uSWQiOiIxX01YNDBOVFU1T1RneU1uNS1NVFEzT1RfMk1qVXpNVEk1Tm40NFpHM
UhaelZzVTJwTGRUSlRaekpWkZWellyb2x1Y3Rpb25fZGF0YT01N0I1MjJlc2VyTmFtZSUyMiUzQSUyMk
Fub255bW91cyUyMFVzZXI3MiUyMiU3RA==", "chromeExtId": "undefined", "sid": "1_M440NTU5OTgyMn5-MTQ3OTE2MjUzMTI5Nn44Z
G1HZzVsU2pLdTJTZzJhZGVzYXo3WUN-fg" }
```

The output has key/value pairs for the API key ("apiKey"), token ("token"), and session ID ("sid"). Save this information. Tokbox monitors the entire flow for failed and successful sessions (only if you were able to successfully verify the Connectivity Doctor ports requirements first). We'll use this metadata to debug your issue. Go to step 4.

Step 4: Open Support Ticket

Please create a [Salesforce support](#) ticket or contact your account team for more help.

Answer the following questions in your support request.

1. TOPOLOGY: What is your network topology? How does data flow through the topology to reach our cloud?
2. ENVIRONMENT: Are you using a virtual environment?
3. PROXY & FIREWALL: What is the type and name of your proxy? What are your firewall restrictions? If you are using a proxy can you execute step 1 and step 2 just through a firewall?
4. PLATFORM & VERSION: What platform are you using? What is the OS version? Which SDK are you using? Which SDK version?
5. DESCRIPTION: Describe the problem you encountered, and the steps you took to try to resolve the problem.
6. LOGS: Can you provide us with any error logs from your side or any other information you deem fit for the problem? Include any relevant network traces or screenshots.
7. CONNECTIVITY DOCTOR: What Connectivity Doctor tests failed? Did you open the required ports and still have issues?
8. JSON METADATA: If applicable, send us the Key/Token/Session information captured in step 3.

Reference Documentation

Reference documentation for the Service SDK.

[ServiceCore Reference Docs](#)

Reference docs that are common to all components of the Service SDK.

[Knowledge Reference Docs](#)

Reference documentation for the Knowledge component of the Service SDK.

[Case Management Reference Docs](#)

Reference documentation for the Case Management component of the Service SDK.

[Live Agent Chat Reference Docs](#)

Reference documentation for the Live Agent Chat component of the Service SDK.

[SOS Reference Docs](#)

Reference documentation for the SOS component of the Service SDK.

ServiceCore Reference Docs

Reference docs that are common to all components of the Service SDK.

[SCAppearanceConfiguration Class Reference](#)

Configuration class used to customize the settings that influence the branding and coloring of UI elements used throughout ServiceCloudKit.

[SCAppearanceConfigurationDelegate Protocol Reference](#)

Delegate protocol for responding to style and branding customization requests at runtime.

[SCServiceCloud Class Reference](#)

Shared singleton class used as the primary access point for ServiceCloudKit.

[SCServiceCloudDelegate Protocol Reference](#)

Delegate protocol to interact with the Service Cloud Kit SDK, and to make decisions on behalf of the SDK.

[SCServiceErrorCode Constants Reference](#)

Possible error response codes when authenticating a user with your Salesforce org.

[SCSNotification Class Reference](#)

A SCSNotification object represents a push notification that was sent to the app from your Salesforce connected app.

[SCSNotificationType Constants Reference](#)

A SCSNotificationType defines the type of notification.

[SCSServiceConfiguration Class Reference](#)

Class used to specify Service Cloud configuration settings.

[ServiceCore String Constants](#)

List of string constants that are shared across all Service SDK frameworks.

SCAppearanceConfiguration Class Reference

Configuration class used to customize the settings that influence the branding and coloring of UI elements used throughout ServiceCloudKit.

Pass an instance of this class to the [appearanceConfiguration](#) property on the [SCServiceCloud](#) shared instance to set the appearance.

delegate

The appearance delegate, used for branding customization.

Declaration

```
@property (nonatomic, weak, nullable) NSObject<SCAppearanceConfigurationDelegate>
*delegate
```

See Also

- [SCAppearanceConfigurationDelegate](#)

Declared In

SCAppearanceConfiguration.h

appearanceApplied

Property indicating whether any `UIAppearance` settings have been applied from this configuration instance.

Declaration

```
@property (nonatomic, readonly, getter=isAppearanceApplied) BOOL appearanceApplied
```

Declared In

SCAppearanceConfiguration.h

– setNeedsAppearanceUpdates

Controls whether the `UIAppearance` selectors need to be updated.

Declaration

```
– (void) setNeedsAppearanceUpdates
```

See Also

- [– applyAppearanceUpdatesIfNeeded](#)

Declared In

SCAppearanceConfiguration.h

– applyAppearanceUpdatesIfNeeded

Updates the `UIAppearance` selectors to reflect the current state of the configuration object.

Declaration

```
– (void) applyAppearanceUpdatesIfNeeded
```

Discussion

This method can be used to force appearance updates to occur at a particular time, or if there is cause for some branding changes to be reapplied. Note that updates are only applied if the appearance settings have indeed changed.

See Also

- [- setNeedsAppearanceUpdates](#)

Declared In

SCAppearanceConfiguration.h

CATEGORY: Fonts Methods**– setFontDescriptor:fontFileName:forWeight:**

Sets `fontDescriptor` for specified weight; used to create and set the `UIFont` on the labels.

Declaration

```
- (void)setFontDescriptor:(UIFontDescriptor *)fontDescriptor fontFileName:(nullable NSString *)fileName forWeight:(NSInteger)weight
```

Parameters

Name	Description
fontDescriptor	<code>UIFontDescriptor</code> instance to be used for creating <code>UIFont</code> .
fileName	Name of the font file contained in the bundle, or <code>nil</code> when using a built-in system font.
weight	A value from the <code>SCFontWeight</code> enum.

Discussion

If `UIFontDescriptorSizeAttribute` is set on the `fontDescriptor`, it is used to set the font size. If not, the SDK sets the font size appropriately.

Declared In

SCAppearanceConfiguration.h

– getFontDescriptorForWeight:

Returns the `fontDescriptor` for the specified weight.

Declaration

```
- (UIFontDescriptor *)getFontDescriptorForWeight:(NSInteger)weight
```

Parameters

Name	Description
weight	A value from the <code>SCFontWeight</code> enum.

Discussion

If no value has been specified, the default value is returned. Default values:

- Helvetica Neue - Light for `SCFontWeightLight`,
- Helvetica Neue for `SCFontWeightRegular`,
- Helvetica Neue - Semibold for `SCFontWeightBold`.

Declared In

`SCAppearanceConfiguration.h`

– `getFontDescriptorForWeight:size:`

Returns the `fontDescriptor` for the specified weight.

Declaration

```
- (UIFontDescriptor *)getFontDescriptorForWeight:(NSInteger)weight size:(CGFloat)size
```

Parameters

Name	Description
weight	A value from the <code>SCFontWeight</code> enum.
size	Size of the desired font

Discussion

If no value has been specified, the default value is returned. Default values iOS 8 & Below:

- Helvetica Neue - Thin for `SCFontWeightLight`,
- Helvetica Neue - Regular for `SCFontWeightRegular`,
- Helvetica Neue - MediumP4 for `SCFontWeightBold`.

Default values iOS 9 & Above:

- San Francisco with `UIFontWeightThin` for `SCFontWeightLight`
- San Francisco with `UIFontWeightRegular` for `SCFontWeightRegular`
- San Francisco with `UIFontWeightSemibold` for `SCFontWeightBold`

Declared In

`SCAppearanceConfiguration.h`

CATEGORY: Images Methods

– setImage:compatibleWithTraitCollection:forName:

Sets the image for a named `SCAppearanceConfiguration` constant.

Declaration

```
- (void)setImage:(UIImage *)image compatibleWithTraitCollection:(nullable UITraitCollection *)traitCollection forName:(NSString *)name
```

Parameters

Name	Description
image	The image to set.
traitCollection	The trait collection the image will be used for, if applicable.
name	The branding token name to set the image for.

Declared In

`SCAppearanceConfiguration.h`

– imageName:compatibleWithTraitCollection:

Returns the image set by the `setImage:forName:` method for the named `SCAppearanceConfiguration` constant.

Declaration

```
- (nullable UIImage *)imageName:(NSString *)name compatibleWithTraitCollection:(nullable UITraitCollection *)traitCollection
```

Parameters

Name	Description
name	The image name to retrieve an image for.
traitCollection	The trait collection the image will be used for, if applicable.

Discussion

If no value has been specified, the default value is returned.

Declared In

`SCAppearanceConfiguration.h`

CATEGORY: NSAttributedString Methods

– setTextAttributes:compatibleWithTraitCollection:forName:

Sets the attributed text attributes for the named `SCAppearanceConfiguration` constant.

Declaration

```
- (void)setTextAttributes:(NSDictionary<NSString*,id> *)attributes  
compatibleWithTraitCollection:(nullable UITraitCollection *)traitCollection  
forName:(NSString *)name
```

Parameters

Name	Description
attributes	The text attributes to set.
traitCollection	The trait collection these attributes will be used for, if applicable.
name	The branding token name to set the attributes for.

Declared In

SCAppearanceConfiguration.h

– textAttributesForName:compatibleWithTraitCollection:

Returns the text attribute set by the `setTextAttributes:compatibleWithTraitCollection:forName:` method for the named `SCAppearanceConfiguration` constant.

Declaration

```
- (nullable NSDictionary<NSString*,id> *)textAttributesForName:(NSString *)name  
compatibleWithTraitCollection:(nullable UITraitCollection *)traitCollection
```

Parameters

Name	Description
name	The branding token name to retrieve the attributes for.
traitCollection	The trait collection these attributes will be used for, if applicable.

Discussion

If no value has been specified, the default value is returned.

Declared In

SCAppearanceConfiguration.h

CATEGORY: Colors Methods

– setColor:forName:

Sets the color for the named `SCAppearanceConfiguration` constant.

Declaration

```
- (void)setColor:(UIColor *)color forName:(NSString *)name
```

Parameters

Name	Description
color	The color to set.
name	The branding token name to set the color for.

Declared In

SCAppearanceConfiguration.h

– colorForName:

Returns the color set by the `setColor:forName:` method for the named `SCAppearanceConfiguration` constant.

Declaration

```
- (nullable UIColor *)colorForName:(NSString *)name
```

Parameters

Name	Description
name	The branding token name to retrieve a color for.

Return Value

The color associated with that branding token, or `nil` if the branding token name is invalid.

Discussion

If no value has been specified, the default value is returned.

Declared In

SCAppearanceConfiguration.h

SCAppearanceConfigurationDelegate Protocol Reference

Delegate protocol for responding to style and branding customization requests at runtime.

– appearanceConfigurationWillApplyUpdates:

Tells the delegate when appearance configuration updates will be applied.

Declaration

```
- (void)appearanceConfigurationWillApplyUpdates:(SCAppearanceConfiguration *)configuration
```

Parameters

Name	Description
configuration	The appearance configuration to be applied.

Declared In

SCAppearanceConfiguration.h

– appearanceConfigurationDidApplyUpdates:

Tells the delegate when appearance configuration updates have been applied.

Declaration

```
- (void)appearanceConfigurationDidApplyUpdates:(SCAppearanceConfiguration *)configuration
```

Parameters

Name	Description
configuration	The appearance configuration that was applied.

Declared In

SCAppearanceConfiguration.h

SCServiceCloud Class Reference

Shared singleton class used as the primary access point for ServiceCloudKit.

Use the [sharedInstance](#) class method to access the singleton for this class. Configure ServiceCloudKit using the [serviceConfiguration](#) property. Customize the appearance of the interface using the [appearanceConfiguration](#) property. See the other properties in this class for additional functionality.

Interface for accessing the [SOSSessionManager](#) instance via the Service Common singleton.

Interface for accessing the [SCSChat](#) instance via the Service Common singleton manager.

+ sharedInstance

Returns the shared service cloud manager for this process.

Declaration

```
+ (instancetype)sharedInstance
```

Return Value

Initialized shared instace.

Declared In

SCServiceCloud.h

primaryWindow

Identifies the primary window where view controllers should be placed when presented.

Declaration

```
@property (null_resettable, nonatomic, strong) UIWindow *primaryWindow
```

Discussion

This value is automatically determined under most circumstances, but in the event that a custom value needs to be set, the `primaryWindow` property can be set to a specific window that should be used.

Declared In

`SCServiceCloud.h`

delegate

The delegate for the service interface manager.

Declaration

```
@property (nonatomic, weak, nullable) NSObject<SCServiceCloudDelegate> *delegate
```

See Also

- [SCServiceCloudDelegate](#)

Declared In

`SCServiceCloud.h`

interfaceStyle

The appearance style to use in presented views and view controllers.

Declaration

```
@property (nonatomic, copy) NSString *interfaceStyle
```

Discussion

The default value for this is `SCInterfaceStyleDefault`. If this value is set after the interface has already been presented, it results in the service interface being dismissed.

WARNING: It is recommended that you only assign this property once in the lifetime of an application. Changing interface styles at runtime may have unpredictable results, and the behavior is undefined.

Declared In

`SCServiceCloud.h`

appearanceConfiguration

Sets the appearance configuration for the service interface. If not specified, this value falls back to default values.

Declaration

```
@property (nonatomic, strong) SCAppearanceConfiguration *appearanceConfiguration
```

See Also

- [SCAppearanceConfiguration](#)

Declared In

`SCServiceCloud.h`

serviceConfiguration

Configuration object used to define the self-service parameters used to enable with Cases and/or Knowledge features.

Declaration

```
@property (nonatomic, strong) SCSServiceConfiguration *serviceConfiguration
```

See Also

- [SCSServiceConfiguration](#)

Declared In

SCServiceCloud.h

account

Mobile SDK user account instance to use for interactions with Salesforce.

Declaration

```
@property (null_resettable, nonatomic, strong) SFUserAccount *account
```

Discussion

This property is automatically reset to a guest user account instance when `nil` is assigned.

Declared In

SCServiceCloud.h

– setAccount:completion:

Sets the [account](#) property asynchronously, allowing authentication errors to be received and processed by the caller.

Declaration

```
- (void)setAccount:(SFUserAccount *)account completion:(nullable void ( ^ ) ( NSError *_Nullable ))completion
```

Parameters

Name	Description
account	User account object to set.
completion	Completion block invoked when the account information has been validated.

Discussion

Note: This setter requires a nonnull [account](#) object to be assigned.

Declared In

SCServiceCloud.h

CATEGORY: SCSNotifications Methods

– notificationFromRemoteNotificationDictionary:

Returns a concrete subclass of [SCSNotification](#) from a remote notification dictionary. Returns `nil` if the notification cannot be handled by the SDK. Use [showInterfaceForNotification:](#) to show the view associated with this notification.

Declaration

```
- (nullable SCSNotification *)notificationFromRemoteNotificationDictionary:(NSDictionary *)userInfo
```

Parameters

Name	Description
userInfo	The remote notification dictionary.

See Also

- – [showInterfaceForNotification:](#)

Declared In

SCServiceCloud.h

– showInterfaceForNotification:

Shows the appropriate view related to the notification that is passed to this method. Use [notificationFromRemoteNotificationDictionary:](#) to convert an `NSDictionary` into an [SCSNotification](#) object.

Declaration

```
- (BOOL)showInterfaceForNotification:(SCSNotification *)notification
```

Parameters

Name	Description
notification	The notification that determines what interface to show.

See Also

- – [notificationFromRemoteNotificationDictionary:](#)

Declared In

SCServiceCloud.h

CATEGORY: SOSSessionManager Methods

SOS

The singleton instance of the [SOSSessionManager](#)

Declaration

```
@property (nonatomic, strong, readonly) SOSSessionManager *sos
```

Declared In

SCServiceCloud+SOSSessionManager.h

CATEGORY: CaseUI Methods**cases**

Case interface configuration and setup.

Declaration

```
@property (nonatomic, strong, readonly) SCCaseInterface *cases
```

See Also

- [SCCaseInterface](#)

Declared In

SCServiceCloud+CaseUI.h

CATEGORY: KnowledgeUI Methods**knowledge**

Knowledge interface configuration and setup.

Declaration

```
@property (nonatomic, strong, readonly) SCKnowledgeInterface *knowledge
```

See Also

- [SCKnowledgeInterface](#)

Declared In

SCServiceCloud+KnowledgeUI.h

CATEGORY: SCSCChat Methods**chat**

The singleton instance of [SCSCChat](#)

Declaration

```
@property (nonatomic, strong, readonly) SCSCChat *chat
```

Declared In

SCServiceCloud+SCSCChat.h

SCServiceCloudDelegate Protocol Reference

Delegate protocol to interact with the Service Cloud Kit SDK, and to make decisions on behalf of the SDK.

– serviceCloud:willDisplayViewController:animated:

Tells the delegate when the interface is about to be displayed.

Declaration

```
- (void)serviceCloud:(SCServiceCloud *)serviceCloud  
willDisplayViewController:(UIViewController *)controller animated:(BOOL)animated
```

Parameters

Name	Description
serviceCloud	Service Cloud interface instance.
controller	View controller being displayed.
animated	YES if the presentation is animated.

Declared In

SCServiceCloudDelegate.h

– serviceCloud:didDisplayViewController:animated:

Calls the delegate after the interface was displayed.

Declaration

```
- (void)serviceCloud:(SCServiceCloud *)serviceCloud  
didDisplayViewController:(UIViewController *)controller animated:(BOOL)animated
```

Parameters

Name	Description
serviceCloud	Service Cloud interface instance.
controller	View controller being displayed.
animated	YES if the presentation was animated.

Declared In

SCServiceCloudDelegate.h

– serviceCloud:transitioningDelegateForViewController:

Asks the delegate to provide a `UIViewControllerTransitioningDelegate` used when a view controller is about to be presented.

Declaration

```
- (nullable NSObject<UIViewControllerTransitioningDelegate> *)serviceCloud:(SCServiceCloud *)serviceCloud transitioningDelegateForViewController:(UIViewController *)controller
```

Parameters

Name	Description
serviceCloud	Service Cloud interface instance.
controller	View controller about to be presented.

Return Value

Transitioning delegate that should control the controller presentation, or `nil` to accept the system defaults.

Discussion

By default, Service Cloud Kit uses custom controllers for presentation and for transition animations. If you wish to customize the default behavior, implement this method and return an object conforming to `UIViewControllerTransitioningDelegate`.

Declared In

`SCServiceCloudDelegate.h`

– serviceCloud:shouldShowActionWithName:

Asks the delegate whether the specified action button should be present.

Declaration

```
- (BOOL)serviceCloud:(SCServiceCloud *)serviceCloud shouldShowActionWithName:(NSString *)name
```

Parameters

Name	Description
serviceCloud	Service Cloud interface instance.
name	Name representing the action in question.

Return Value

`YES` if the action button should be visible; `NO` if the action button should be omitted.

Discussion

The value of the action name can be one of these constants (defined in Service Cloud Kit):

- `SCSActionCasePublisher`: for the case publisher action
- `SCSActionCaseList`: for the case list action
- `SCSActionArticleSearch`: for the knowledge article search action

Declared In`SCServiceCloudDelegate.h`**– serviceCloud:shouldAuthenticateService:completion:**

Asks the delegate whether authentication should be performed for the specified service.

Declaration

```
- (BOOL)serviceCloud:(SCServiceCloud *)serviceCloud shouldAuthenticateService:(NSString *)service completion:(void ( ^ ) ( SFUserAccount *_Nullable account ))completion
```

Parameters

Name	Description
serviceCloud	Service Cloud interface instance.
service	The associated service.
completion	The completion block you should call when you've retrieved the user account.

Return Value

`YES` if the receiver plans to supply authentication information for this service, otherwise `NO` if the current account credentials are sufficient.

Discussion

The value of the service name can be one of these constants (defined in Service Cloud Kit):

- `SCServiceTypeCases`: for the Case Management service
- `SCServiceTypeKnowledge`: for the Knowledge service

If the delegate returns `YES` to this method, it is the responsibility of the receiver to supply the new user account object to the supplied completion block. The value supplied to the completion block will replace the [account](#) property.

Declared In`SCServiceCloudDelegate.h`**– serviceCloud:serviceAuthenticationFailedWithError:**

Tells the delegate when there is an error in authenticating Case Management or Knowledge.

Declaration

```
- (void)serviceCloud:(SCServiceCloud *)serviceCloud serviceAuthenticationFailedWithError:(NSError *)error
```

Parameters

Name	Description
serviceCloud	Service Cloud interface instance.

Name	Description
error	The error message associated with the failure.

Declared In

SCServiceCloudDelegate.h

SCServiceErrorCode Constants Reference

Possible error response codes when authenticating a user with your Salesforce org.

Definition

```
typedef NS_ENUM(NSInteger, SCServiceErrorCode ) {

    SCServiceUserSessionExpiredOrInvalidError = 401,

    SCServiceUserRequestRefusedError = 403,

    SCServiceUserResourceNotFoundError = 404,

};
```

Constants**SCServiceUserSessionExpiredOrInvalidError**

Error condition when the session has expired or is invalid.

SCServiceUserRequestRefusedError

Error condition when the request to authenticate a user is refused by the server.

SCServiceUserResourceNotFoundError

Error condition when the requested resource is not found on the server.

Declared In

SCCDefines.h

SCSNotification Class Reference

A SCSNotification object represents a push notification that was sent to the app from your Salesforce connected app.

SCSNotification class itself is an abstract class. Based on the userInfo dictionary passed to it from the push notification payload, a concrete subclass of SCSNotification is returned by calling: [\[SCServiceCloud notificationFromRemoteNotificationDictionary:\]](#)

notificationType

Use this property to find out the type of SCSNotification instead of using isKindOfClass.

Declaration

```
@property (nonatomic, readonly, assign) SCSNotificationType notificationType
```

Declared In

SCSNotification.h

SCSNotificationType Constants Reference

A SCSNotificationType defines the type of notification.

Definition

```
typedef NS_ENUM(NSInteger, SCSNotificationType ) {  
  
    SCSNotificationTypeCase = 1,  
  
};
```

Constants**SCSNotificationTypeCase**

Notification associated with a case.

Declared In

SCSNotification.h

SCSServiceConfiguration Class Reference

Class used to specify Service Cloud configuration settings.

Pass an instance of this class to the [serviceConfiguration](#) property on the [SCServiceCloud](#) shared instance to set the configuration.

communityURL

The URL of the community that hosts the public knowledge base to be exposed.

Declaration

```
@property (nonatomic, strong, readonly) NSURL *communityURL
```

See Also

- - [initWithCommunity:](#)

Declared In

SCSServiceConfiguration.h

– initWithCommunity:

Creates a service configuration with the given [communityURL](#).

Declaration

```
- (instancetype) initWithCommunity: (NSURL *) communityURL
```

Parameters

Name	Description
communityURL	URL of the community which hosts the public knowledge base.

Declared In

SCSServiceConfiguration.h

CATEGORY: KnowledgeCore Methods**dataCategoryGroup**

The unique name of the data category group to use when communicating with Service Cloud.

Declaration

```
@property (nullable, nonatomic, copy, readonly) NSString *dataCategoryGroup
```

See Also

- [@property rootDataCategory](#)
- [- initWithCommunity:dataCategoryGroup:rootDataCategory:](#)

Declared In

SCSServiceConfiguration+KnowledgeCore.h

rootDataCategory

The data category to use as the root when displaying knowledge articles.

Declaration

```
@property (nullable, nonatomic, copy, readonly) NSString *rootDataCategory
```

See Also

- [@property dataCategoryGroup](#)
- [- initWithCommunity:dataCategoryGroup:rootDataCategory:](#)

Declared In

SCSServiceConfiguration+KnowledgeCore.h

language

The language to use when loading knowledge articles.

Declaration

```
@property (nullable, nonatomic, copy) NSString *language
```

Declared In

```
SCSServiceConfiguration+KnowledgeCore.h
```

imageFolderPath

The path pointing to the folder where the images in Data Category Header View, Article Header View, and Article Cell are rendered from. This can be an absolute path or relative path to the application bundle.

Declaration

```
@property (nullable, nonatomic, copy) NSString *imageFolderPath
```

Declared In

```
SCSServiceConfiguration+KnowledgeCore.h
```

articleSortByField

Specifies the sort by field to be used to sort while displaying articles in Support home and Data Category Detail and article list screen

Declaration

```
@property (nonatomic) SCArticleSortByField articleSortByField
```

Declared In

```
SCSServiceConfiguration+KnowledgeCore.h
```

articleSortOrder

Specifies the sort order (ascending or descending) to be used to sort while displaying articles in Support home and Data Category Detail and article list screen

Declaration

```
@property (nonatomic) SCArticleSortOrder articleSortOrder
```

Declared In

```
SCSServiceConfiguration+KnowledgeCore.h
```

– initWithCommunity:dataCategoryGroup:rootDataCategory:

Creates a service configuration with the given [communityURL](#), data category group, and root data category.

Declaration

```
- (instancetype) initWithCommunity: (NSURL *) communityURL dataCategoryGroup: (NSString *) dataCategoryGroup rootDataCategory: (NSString *) rootDataCategory
```

Parameters

Name	Description
communityURL	URL of the community which hosts the public knowledge base.
dataCategoryGroup	Data category group name.
rootDataCategory	Root data category name.

Declared In

SCSServiceConfiguration+KnowledgeCore.h

ServiceCore String Constants

List of string constants that are shared across all Service SDK frameworks.

Table 9: String Constants

String Token	Description	Default Value
ServiceCloud.Common.NoConnectionErrorMsgTitle	Title displayed when there is no internet connection	Unable to connect...
ServiceCloud.Common.NoConnectionErrorMsgDescription	Description displayed when there is no internet connection	Please ensure you are online
ServiceCloud.Common.Error.UserSessionExpiredOrInvalidSession	Description displayed when user account has expired	The session ID or OAuth token used has expired or is invalid.
ServiceCloud.Common.Error.UserSessionRequestRefused	Description displayed when user does not have sufficient access.	The request has been refused. Verify that the logged-in user has appropriate permissions.
ServiceCloud.Common.Error.ResourceNotFound	Description displayed when there is an Authentication Failure	The requested resource could not be found.
ServiceCloud.Common.Error.GenericErrorMsgTitle	Main Message displayed for generic error	Oops. Something went wrong...
ServiceCloud.Common.Error.GenericErrorMsgDescription	Second line of Message displayed for generic error condition	Please try again or contact the administrator

Knowledge Reference Docs

Reference documentation for the Knowledge component of the Service SDK.

[SCArticleSortByField Constants Reference](#)

Field to sort articles by.

[SCArticleSortOrder Constants Reference](#)

Article sort order.

[SCChannelType Constants Reference](#)

Knowledge channel type.

[SCKnowledgeInterface Class Reference](#)

Primary access point when interacting with Knowledge.

[SCKnowledgeInterfaceDelegate Protocol Reference](#)

Delegate protocol that defines the methods sent to the Knowledge delegate.

[SCQueryMethod Constants Reference](#)

Knowledge article query method.

[SCSArticle Class Reference](#)

A SCSArticle object represents a knowledge base article.

[SCSArticleDownloadOption Constants Reference](#)

Download options used when fetching article content.

[SCSArticleQuery Class Reference](#)

An SCSArticleQuery object manages the criteria to apply when fetching and searching knowledge articles. This query object stores the type of search, the search parameters, and any filters or constraints to apply when searching.

[SCSArticleViewController Class Reference](#)

View controller capable of displaying the contents of an article. When used in conjunction with a navigation controller, the title property of the view will automatically be set to the title of the article, or nil if no article is assigned.

[SCSArticleViewControllerDelegate Protocol Reference](#)

Delegate protocol used to interact with article view controllers.

[SCSCategory Class Reference](#)

A SCSCategory represents an individual data category within a data category group. Categories form a tree hierarchy, where an individual category has one parent category (with the exception of a root-level category, which does not have a parent), and it may have multiple child categories.

[SCSCategoryGroup Class Reference](#)

A SCSCategoryGroup represents a data category group. A category group contains a hierarchical tree of data categories.

[SCSKnowledgeErrorCode Constants Reference](#)

Possible Knowledge error response codes.

[SCSKnowledgeManager Class Reference](#)

A SCSKnowledgeManager object is used to interact with knowledge data such as category groups, categories, and articles.

[SCSMutableArticleQuery Class Reference](#)

An SCSMutableArticleQuery object is the mutable version of SCSArticleQuery. It manages the criteria to apply when fetching and searching knowledge articles. This query object stores the type of search, the search parameters, and any filters or constraints to apply when searching.

[KnowledgeFramework String Constants](#)

List of string constants associated with the Service SDK Knowledge framework.

SCArticleSortByField Constants Reference

Field to sort articles by.

Definition

```
typedef NS_ENUM(NSInteger, SCArticleSortByField ) {  
  
    SCArticleSortByFieldUnknown,  
  
    SCArticleSortByFieldLastPublishedDate,  
  
    SCArticleSortByFieldTitle,  
  
    SCArticleSortByFieldViewScore,  
  
};
```

Constants

SCArticleSortByFieldUnknown

Unknown sort field.

SCArticleSortByFieldLastPublishedDate

Sort by last published date.

SCArticleSortByFieldTitle

Sort by title field.

SCArticleSortByFieldViewScore

Sort by view score field.

Declared In

SCDefines.h

SCArticleSortOrder Constants Reference

Article sort order.

Definition

```
typedef NS_ENUM(NSInteger, SCArticleSortOrder ) {  
  
    SCArticleSortOrderUnknown,  
  
    SCArticleSortOrderAscending,  
  
    SCArticleSortOrderDescending,  
  
};
```

Constants

SCArticleSortOrderUnknown

Unknown sort order.

SCArticleSortOrderAscending

Sort in an ascending order.

SCArticleSortOrderDescending

Sort in a descending order.

Declared In

SCDefines.h

SCChannelType Constants Reference

Knowledge channel type.

Definition

```
typedef NS_ENUM(NSInteger, SCChannelType ) {  
  
    SCChannelTypeUnknown,  
  
    SCChannelTypeInternal,  
  
    SCChannelTypePublic,  
  
    SCChannelTypeCustomerPortal,  
  
    SCChannelTypePartnerPortal,  
  
};
```

Constants

SCChannelTypeUnknown

Unknown channel.

SCChannelTypeInternal

Internal channel.

SCChannelTypePublic

Public channel.

SCChannelTypeCustomerPortal

Customer portal channel.

SCChannelTypePartnerPortal

```
Partner portal channel.
```

Declared In

```
SCDefines.h
```

SCKnowledgeInterface Class Reference

Primary access point when interacting with Knowledge.

Use this class to configure and customize the Knowledge interface. Get an instance of this class with the [knowledge](#) property on the [SCServiceCloud](#) shared instance.

enabled

Indicates whether the Knowledge interface is enabled.

Declaration

```
@property (nonatomic, assign, getter=isEnabled) BOOL enabled
```

Declared In

```
SCKnowledgeInterface.h
```

interfaceVisible

Indicates whether the Knowledge interface is currently visible. Setting this value will present or dismiss the interface in an unanimated fashion.

Declaration

```
@property (nonatomic, assign, getter=isInterfaceVisible) BOOL interfaceVisible
```

See Also

- - [setInterfaceVisible:animated:completion:](#)

Declared In

```
SCKnowledgeInterface.h
```

delegate

The delegate for the Knowledge interface.

Declaration

```
@property (nonatomic, weak) NSObject<SCKnowledgeInterfaceDelegate> *delegate
```

See Also

- [SCKnowledgeInterfaceDelegate](#)

Declared In

```
SCKnowledgeInterface.h
```

– setInterfaceVisible:animated:completion:

Controls the visibility of the Knowledge interface, with an optional animation.

Declaration

```
- (void)setInterfaceVisible:(BOOL) interfaceVisible animated:(BOOL) animated
completion:(dispatch_block_t) completionBlock
```

Parameters

Name	Description
<code>interfaceVisible</code>	YES to make the interface visible; NO to hide the interface.
<code>animated</code>	YES to animate; NO to perform no animation.
<code>completionBlock</code>	Optional block to invoke when the interface change has completed.

Declared In

`SCKnowledgeInterface.h`

remoteLoggingEnabled

Determines whether session logs are sent for collection. Logs sent remotely do not collect personal information. Unique IDs are created for tying logs to sessions, and those IDs cannot be correlated back to specific users.

Declaration

```
@property (nonatomic) BOOL remoteLoggingEnabled
```

Discussion

Default: YES

Declared In

`SCKnowledgeInterface.h`

SCKnowledgeInterfaceDelegate Protocol Reference

Delegate protocol that defines the methods sent to the Knowledge delegate.

– knowledgeInterface:imageForArticle:compatibleWithTraitCollection:

Asks the delegate for an article image. Use this method to customize the image for an article header and cell.

Declaration

```
- (nullable UIImage *)knowledgeInterface:(SCKnowledgeInterface *) interface
imageForArticle:(NSString *) articleId compatibleWithTraitCollection:(nullable
UITraitCollection *) traitCollection
```


Parameters

Name	Description
interface	The Knowledge interface instance.
articleId	The unique article number.
traitCollection	Traits that describe the desired image to retrieve.

Declared In

SCKnowledgeInterfaceDelegate.h

– knowledgeInterface:imageForDataCategory:compatibleWithTraitCollection:

Asks the delegate for a data category image. Use this method to customize the image for a data category header and cell.

Declaration

```
- (nullable UIImage *)knowledgeInterface:(SCKnowledgeInterface *)interface  
imageForDataCategory:(NSString *)categoryName compatibleWithTraitCollection:(nullable  
UITraitCollection *)traitCollection
```

Parameters

Name	Description
interface	The Knowledge interface instance.
categoryName	The unique developerName of that category.
traitCollection	Traits that describe the desired image to retrieve.

Declared In

SCKnowledgeInterfaceDelegate.h

SCQueryMethod Constants Reference

Knowledge article query method.

Definition

```
typedef NS_ENUM(NSInteger, SCQueryMethod) {  
  
    SCQueryMethodUnknown,  
  
    SCQueryMethodAbove,  
  
    SCQueryMethodBelow,  
  
    SCQueryMethodAboveOrBelow,  
}
```

```
SCQueryMethodAt,
};
```

Constants

SCQueryMethodUnknown

Unknown query method.

SCQueryMethodAbove

Query above the specified category.

SCQueryMethodBelow

Query below the specified category.

SCQueryMethodAboveOrBelow

Query above or below the specified category.

SCQueryMethodAt

Query within the specified category.

Declared In

SCDefines.h

SCSArticle Class Reference

A SCSArticle object represents a knowledge base article.

This article may or may not already be downloaded locally. Use the [isArticleContentDownloaded](#) property to check whether it is downloaded. You can download the article with the [downloadContentWithOptions:completion:](#) method.

articleId

Unique article ID.

Declaration

```
@property (nonatomic, copy, readonly) NSString *articleId
```

Declared In

SCSArticle.h

articleNumber

Article number.

Declaration

```
@property (nonatomic, copy, readonly) NSString *articleNumber
```

Declared In

SCSArticle.h

summary

Summary text for the article.

Declaration

```
@property (nonatomic, copy, readonly) NSString *summary
```

Declared In

SCSArticle.h

title

Article title.

Declaration

```
@property (nonatomic, copy, readonly) NSString *title
```

Declared In

SCSArticle.h

lastPublishedDate

Last published date for this article.

Declaration

```
@property (nonatomic, strong, readonly) NSDate *lastPublishedDate
```

Declared In

SCSArticle.h

storageUsed

Amount of local storage (in bytes) used for this article.

Declaration

```
@property (nonatomic, assign, readonly) NSUInteger storageUsed
```

Declared In

SCSArticle.h

lastAccessedDate

Date article was last accessed.

Declaration

```
@property (nonatomic, copy, readonly) NSDate *lastAccessedDate
```

Declared In

SCSArticle.h

manager

Knowledge manager instance this article belongs to

Declaration

```
@property (nonatomic, weak, readonly) SCSKnowledgeManager *manager
```

Declared In

SCSArticle.h

stale

Returns true if the article details stored in local storage gets stale

Declaration

```
@property (nonatomic, readonly, getter=isStale) BOOL stale
```

Declared In

SCSArticle.h

– downloadContentWithOptions:completion:

Downloads article contents.

Declaration

```
- (void)downloadContentWithOptions:(SCSArticleDownloadOption)options completion:(nullable void (^) (NSError *__nullable error))completionBlock
```

Parameters

Name	Description
options	Cache options defining exactly what content to download. See SCSArticleDownloadOption .
completionBlock	The block to execute after download completes. You may specify nil for this parameter.

See Also

- [SCSArticleDownloadOption](#)

Declared In

SCSArticle.h

– isArticleContentDownloaded

Whether article content has been downloaded.

Declaration

```
- (BOOL)isArticleContentDownloaded
```

Declared In

SCSArticle.h

– isAssociatedContentDownloaded

Whether associated article content (such as images) have been downloaded

Declaration

```
- (BOOL)isAssociatedContentDownloaded
```

Declared In

SCSArticle.h

SCSArticleDownloadOption Constants Reference

Download options used when fetching article content.

See [SCSArticle](#) for more info.

Definition

```
typedef NS_ENUM(NSInteger, SCSArticleDownloadOption ) {  
  
    SCSArticleDownloadOptionImages = 1 << 0,  
  
    SCSArticleDownloadOptionRefetchArticleContent = 1 << 1,  
  
};
```

Constants**SCSArticleDownloadOptionImages**

Include images when downloading article.

SCSArticleDownloadOptionRefetchArticleContent

Refetch article fields from server when downloading.

Declared In

SCDefines.h

SCSArticleQuery Class Reference

An `SCSArticleQuery` object manages the criteria to apply when fetching and searching knowledge articles. This query object stores the type of search, the search parameters, and any filters or constraints to apply when searching.

This class is immutable. To create a mutable query object, see [SCSMutableArticleQuery](#).

articleId

Specifies the 18-character article ID. This property cannot be used with [searchTerm](#), [queryMethod](#), [sortOrder](#), or [sortByField](#).

Declaration

```
@property (nullable, nonatomic, copy, readonly) NSString *articleId
```

Declared In

SCSArticleQuery.h

categories

Specifies the data categories associated with the articles that need to be queried.

Declaration

```
@property (nullable, nonatomic, copy, readonly) NSArray<SCSCategory*> *categories
```

Discussion

Use this property with the [queryMethod](#) property to further refine the search. When specifying multiple categories, the category group must be different for each element.

See Also

- [SCSCategory](#)

Declared In

SCSArticleQuery.h

queryMethod

If the [categories](#) property is populated, this property determines the method used to select articles above, at, below, or above_or_below the specified [categories](#).

Declaration

```
@property (nonatomic, assign, readonly) SCQueryMethod queryMethod
```

See Also

- [SCQueryMethod](#)

Declared In

SCSArticleQuery.h

searchTerm

Specifies the search term to be used to query articles. This property cannot be used with [articleId](#), [sortOrder](#), or [sortByField](#).

Declaration

```
@property (nullable, nonatomic, copy, readonly) NSString *searchTerm
```

Declared In

SCSArticleQuery.h

pageSize

Specifies the number of articles to fetch.

Declaration

```
@property (nonatomic, assign, readonly) NSUInteger pageSize
```

Discussion

An org does not return more than 100 results per page.

Declared In

SCSArticleQuery.h

sortOrder

If the [categories](#) property is populated, this property specifies the sort order to be used.

Declaration

```
@property (nonatomic, assign, readonly) SCArticleSortOrder sortOrder
```

See Also

- [SCArticleSortOrder](#)

Declared In

SCSArticleQuery.h

sortByField

If the [categories](#) property is populated, this property specifies the article field used for sorting the article list.

Declaration

```
@property (nonatomic, assign, readonly) SCArticleSortByField sortByField
```

See Also

- [SCArticleSortByField](#)

Declared In

SCSArticleQuery.h

valid

Whether the query criteria is valid.

Declaration

```
@property (nonatomic, readonly, getter=isValid) BOOL valid
```

Discussion

If the query is not valid, fetches and article lookups cannot be performed with this object.

Declared In

SCSArticleQuery.h

SCSArticleViewController Class Reference

View controller capable of displaying the contents of an article. When used in conjunction with a navigation controller, the title property of the view will automatically be set to the title of the article, or nil if no article is assigned.

category

The category information to use when displaying empty content. If nil is assigned, the value will automatically reset to the root data category.

Declaration

```
@property (null_resettable, nonatomic, strong) SCSCategory *category
```

Declared In

SCSArticleViewController.h

article

The article to display. If the article is not yet cached, its content will automatically be downloaded on-demand when the view is loaded. If nil is assigned, an empty view will automatically be shown to display information about the data [category](#).

Declaration

```
@property (nullable, nonatomic, copy) SCSArticle *article
```

Declared In

SCSArticleViewController.h

delegate

Delegate property to use to interact with the [article](#) view controller.

Declaration

```
@property (nullable, nonatomic, weak) NSObject<SCSArticleViewControllerDelegate> *delegate
```

Declared In

SCSArticleViewController.h

SCSArticleViewControllerDelegate Protocol Reference

Delegate protocol used to interact with article view controllers.

– articleController:categoryDidChange:

Message sent to the delegate when the selected category is changed.

Declaration

```
- (void)articleController:(SCSArticleViewController *)controller  
categoryDidChange:(SCSCategory *)newCategory
```


Parameters

Name	Description
controller	The article view controller.
category	The new category.

Declared In

SCSArticleViewControllerDelegate.h

– articleController:willShowArticle:

Informs the delegate when an article will be shown.

Declaration

```
- (void)articleController:(SCSArticleViewController *)controller willShowArticle:(nullable SCSArticle *)article
```

Parameters

Name	Description
controller	The article view controller.
article	The article to be displayed, or <code>nil</code> if no article is selected.

Discussion

This message is sent immediately upon setting a new article.

Declared In

SCSArticleViewControllerDelegate.h

– articleController:didShowArticle:

Informs the delegate when an article has finished being shown.

Declaration

```
- (void)articleController:(SCSArticleViewController *)controller didShowArticle:(nullable SCSArticle *)article
```

Parameters

Name	Description
controller	The article view controller.
article	The article being shown, or <code>nil</code> if no article is selected.

Discussion

This is sent when the article has finished loading and is displayed on the screen. If the view controller isn't initially presented in the view hierarchy when the article is initially set, this message may be delayed until the controller is eventually presented and the article is loaded.

Declared In

SCSArticleViewControllerDelegate.h

– articleController:heightForHeaderView:

The target height for the given header view.

Declaration

```
- (CGFloat)articleController:(SCSArticleViewController *)controller  
heightForHeaderView:(UIView *)headerView
```

Parameters

Name	Description
controller	The article view controller.
headerView	The header view to be displayed.

Return Value

Positive float value for the desired target header view height.

Discussion

If this method isn't implemented, the height for the article header will be determined automatically. If a header view isn't to be used for this article, this method is never called.

Declared In

SCSArticleViewControllerDelegate.h

– articleController:headerViewForArticle:

Header view to display above the article.

Declaration

```
- (nullable UIView *)articleController:(SCSArticleViewController *)controller  
headerViewForArticle:(SCSArticle *)article
```

Parameters

Name	Description
controller	The article view controller.
article	The article being displayed.

Return Value

UIView instance to be used within the header, or `nil`.

Discussion

If this method isn't implemented, a default header view will be used if an image is available for this article. If you do not wish for a header to be displayed, you can return `nil` from this method to opt out of displaying a header.

Auto Layout rules will be applied when adding this view to the hierarchy.

Declared In

`SCSArticleViewControllerDelegate.h`

SCSCategory Class Reference

A `SCSCategory` represents an individual data category within a data category group. Categories form a tree hierarchy, where an individual category has one parent category (with the exception of a root-level category, which does not have a parent), and it may have multiple child categories.

Refer to [SCSCategoryGroup](#) for a representation of a data category group.

name

Unique name for this category.

Declaration

```
@property (nonatomic, copy, readonly) NSString *name
```

Declared In

`SCSCategory.h`

label

Label, or display [name](#), for this category.

Declaration

```
@property (nonatomic, copy, readonly) NSString *label
```

Declared In

`SCSCategory.h`

url

API URL for this category.

Declaration

```
@property (nonatomic, copy, readonly) NSString *url
```

Declared In

`SCSCategory.h`

childCategories

Array of immediate child categories below this one. If this category is a leaf category, this array will be empty.

Declaration

```
@property (nonatomic, strong, readonly) NSArray<SCSCategory*> *childCategories
```

Declared In

SCSCategory.h

parentCategory

Reference to this category's immediate parent. For root-level categories, this value is `nil`.

Declaration

```
@property (nonatomic, weak, readonly, nullable) SCSCategory *parentCategory
```

Declared In

SCSCategory.h

categoryGroup

The category group that this category resides within.

Declaration

```
@property (nonatomic, weak, readonly, nullable) SCSCategoryGroup *categoryGroup
```

See Also

- [SCSCategoryGroup](#)

Declared In

SCSCategory.h

manager

The knowledge manager this category belongs to.

Declaration

```
@property (nonatomic, weak, readonly) SCSKnowledgeManager *manager
```

Declared In

SCSCategory.h

– categoryName:

Returns a category at or below this category that matches the specified [name](#).

Declaration

```
- (nullable SCSCategory *)categoryWithName:(NSString *)name
```

Parameters

Name	Description
name	Category name .

Return Value

SCSCategory instance matching the specified category [name](#), or `nil` if no child category is found.

See Also

- [\[SCSCategoryGroup categoryName:\]](#)

Declared In

SCSCategory.h

SCSCategoryGroup Class Reference

A SCSCategoryGroup represents a data category group. A category group contains a hierarchical tree of data categories.

Refer to [SCSCategory](#) for a representation of data categories.

name

Unique category group name.

Declaration

```
@property (nonatomic, copy, readonly) NSString *name
```

Declared In

SCSCategoryGroup.h

label

Label, or display [name](#), for this category group.

Declaration

```
@property (nonatomic, copy, readonly) NSString *label
```

Declared In

SCSCategoryGroup.h

objectUsage

String describing which types of objects this category group can be used for.

Declaration

```
@property (nonatomic, copy, readonly) NSString *objectUsage
```

Declared In

SCSCategoryGroup.h

childCategories

Array of root-level child categories within this category group.

Declaration

```
@property (nonatomic, strong, readonly) NSArray<SCSCategory*> *childCategories
```

See Also

- [SCSCategory](#)

Declared In

SCSCategoryGroup.h

manager

The knowledge manager this category group belongs to.

Declaration

```
@property (nonatomic, weak, readonly) SCSCKnowledgeManager *manager
```

Declared In

SCSCategoryGroup.h

– categoryName:

Returns a category within this category group that matches the specified [name](#).

Declaration

```
- (nullable SCSCategory *)categoryWithName:(NSString *)name
```

Parameters

Name	Description
name	Category name .

Return Value

[SCSCategory](#) instance matching the specified category [name](#), or `nil` if no child category is found.

See Also

- [\[SCSCategory categoryName:\]](#)

Declared In

SCSCategoryGroup.h

debugDescription

Returns a string suitable for debugging that displays the contents of the category group tree.

Declaration

```
@property (readonly, copy) NSString *debugDescription
```

Declared In`SCSCategoryGroup.h`

SCSKnowledgeErrorCode Constants Reference

Possible Knowledge error response codes.

Definition

```
typedef NS_ENUM(NSInteger, SCSKnowledgeErrorCode ) {  
  
    SCSKnowledgeErrorCodeNoArticleFound = 1,  
  
    SCSKnowledgeErrorCodeCannotWriteFile,  
  
    SCSKnowledgeErrorCodeDataNotFound,  
  
    SCSKnowledgeErrorCodeInvalidCategory,  
  
    SCSKnowledgeErrorCodeInvalidCategoryGroup,  
  
};
```

Constants**SCSKnowledgeErrorCodeNoArticleFound**

No article found.

SCSKnowledgeErrorCodeCannotWriteFile

Unable to write file.

SCSKnowledgeErrorCodeDataNotFound

Data not found.

SCSKnowledgeErrorCodeInvalidCategory

Invalid data category

SCSKnowledgeErrorCodeInvalidCategoryGroup

Invalid data category group

Declared In`SCDefines.h`

SCSKnowledgeManager Class Reference

A `SCSKnowledgeManager` object is used to interact with knowledge data such as category groups, categories, and articles.

The knowledge manager abstracts the storage, network retrieval, and queries for interacting with data in an efficient fashion. You can use this object to fetch categories and articles from your org. Once fetched, you can use this object to get cached content. See the [SCSArticleQuery](#) class for an explanation on how to query for content.

The knowledge manager caches data for offline use, allowing categories and articles to be used even when no network is available. To stay informed about important changes in the underlying data model, any changes to the underlying categories will be advertised through the use of the following notifications:

- `SCSKnowledgeManagerCategoryAddedNotification` – A new category is added.
- `SCSKnowledgeManagerCategoryUpdatedNotification` – A category has been modified (e.g. title, child categories, etc)
- `SCSKnowledgeManagerCategoryDeletedNotification` – A category has been deleted.

The object of the notification will be the `SCSKnowledgeManager` instance itself, while the category in question is stored in the notification's `userInfo` dictionary under the `SCSKnowledgeManagerCategoryKey` key.

userAccount

User account used for network interactions and offline caching.

Declaration

```
@property (nonatomic, strong, readonly) SFUserAccount *userAccount
```

Discussion

This value will change if the `defaultManager` instance is used.

See Also

- [+ defaultManager](#)

Declared In

`SCSKnowledgeManager.h`

+ defaultManager

Singleton instance of the knowledge manager.

Declaration

```
+ (instancetype)defaultManager
```

Return Value

Shared knowledge manager instance.

Discussion

This singleton instance automatically uses the user account currently selected within `[SCServiceCloud userAccount]`.

Declared In

`SCSKnowledgeManager.h`

+ knowledgeManagerWithUserAccount:

Returns a knowledge manager instance with the given user account.

Declaration

```
+ (SCSKnowledgeManager *)knowledgeManagerWithUserAccount:(SFUserAccount *)account
```

Parameters

Name	Description
userAccount	User account to use.

Return Value

Initialized knowledge manager.

Declared In

SCSKnowledgeManager.h

– fetchAllCategoriesWithCompletionHandler:

Triggers a fetch to load the categories for the user's organization.

Declaration

```
- (void)fetchAllCategoriesWithCompletionHandler:(nullable void ( ^ ) (
    NSArray<SCSCategoryGroup*> *__nullable categoryGroups , NSError *__nullable error
)) completionHandler
```

Parameters

Name	Description
completionHandler	Optional completion block to be called when the operation completes.

Discussion

The data returned will automatically be cached, and any changes that have occurred since the previous fetch will be updated. The object of these notifications are

Declared In

SCSKnowledgeManager.h

– hasFetchedCategories

Indicates whether this knowledge manager has fetched new category information since it was created.

Declaration

```
- (BOOL)hasFetchedCategories
```

Return Value

YES if this instance has successfully completed `fetchAllCategoriesWithCompletionHandler:`, otherwise NO.

Discussion

Note: It is not always necessary to fetch new categories, especially since that may be a costly operation to perform if your organization has a large number of categories.

Declared In

`SCSKnowledgeManager.h`

– allCategoryGroups

Returns all the public category groups for the current organization.

Declaration

```
- (NSArray<SCSCategoryGroup*> *)allCategoryGroups
```

Return Value

Array of [SCSCategoryGroup](#) instances

Discussion

Warning: This array may be empty if no data has been previously cached.

See Also

- [SCSCategoryGroup](#)

Declared In

`SCSKnowledgeManager.h`

– categoryGroupWithName:

Returns the category group matching the given name.

Declaration

```
- (nullable SCSCategoryGroup *)categoryGroupWithName:(NSString *)name
```

Parameters

Name	Description
name	Name of the category group to return.

Return Value

[SCSCategoryGroup](#) instance, or `nil` if none is found with that name.

Discussion

Warning: This may return `nil` if no data has been previously cached.

See Also

- [SCSCategoryGroup](#)

Declared In

`SCSKnowledgeManager.h`

– `articleWithId:`

Returns a cached article object with the given article ID.

Declaration

```
- (nullable SCSArticle *)articleWithId:(NSString *)articleId
```

Parameters

Name	Description
<code>articleId</code>	Article ID to search for.

Return Value

Article matching the given ID, or `nil` if no article is found in the cache with that name.

See Also

- [SCSArticle](#)

Declared In

`SCSKnowledgeManager.h`

– `articlesMatchingQuery:completion:`

Returns cached article objects matching the given query.

Declaration

```
- (void)articlesMatchingQuery:(SCSArticleQuery *)query completion:(void ( ^ ) (
    NSArray<SCSArticle*> *_Nullable articles , NSError *_Nullable error ))completion
```

Parameters

Name	Description
<code>query</code>	Query to use for identifying articles.
<code>completion</code>	Completion block to invoke when the articles are found.

Discussion

This method searches through the database of existing article information stored locally to identify articles. No network operations are invoked through this method, and this can safely be used even when offline. For performing requests against the network, please see the [fetchArticlesWithQuery:completion:](#) method

See Also

- [- fetchArticlesWithQuery:completion:](#)
- [SCSArticleQuery](#)
- [SCSArticle](#)

Declared In

`SCSKnowledgeManager.h`

– fetchArticlesWithQuery:completion:

Fetches articles from the server that match the given query.

Declaration

```
- (void)fetchArticlesWithQuery:(SCSArticleQuery *)query completion:(nullable void ( ^ ) ( NSArray<SCSArticle*> *articles , NSError *__nullable error ))completion
```

Parameters

Name	Description
query	Query to use for identifying articles.
completion	Completion block to invoke when articles are found.

See Also

- [SCSArticleQuery](#)
- [SCSArticle](#)

Declared In

SCSKnowledgeManager.h

SCSMutableArticleQuery Class Reference

An `SCSMutableArticleQuery` object is the mutable version of `SCSArticleQuery`. It manages the criteria to apply when fetching and searching knowledge articles. This query object stores the type of search, the search parameters, and any filters or constraints to apply when searching.

Use this object with [\[SCSKnowledgeManager fetchArticlesWithQuery:completion:\]](#) to download articles that match your query. To access already downloaded articles matching your query, call [\[SCSKnowledgeManager articlesMatchingQuery:completion:\]](#).

articleId

Specifies the 18-character article ID. This property cannot be used with [searchTerm](#), [queryMethod](#), [sortOrder](#), or [sortByField](#).

Declaration

```
@property (nullable, nonatomic, copy, readwrite) NSString *articleId
```

Declared In

SCSArticleQuery.h

categories

Specifies the data categories associated with the articles that need to be queried.

Declaration

```
@property (nullable, nonatomic, copy, readwrite) NSArray<SCSCategory*> *categories
```

Discussion

Use this property with the [queryMethod](#) property to further refine the search. When specifying multiple categories, the category group must be different for each element.

See Also

- [SCSCategory](#)

Declared In

SCSArticleQuery.h

queryMethod

If the [categories](#) property is populated, this property determines the method used to select articles above, at, below, or above_or_below the specified [categories](#).

Declaration

```
@property (nonatomic, assign, readwrite) SCQueryMethod queryMethod
```

See Also

- [SCQueryMethod](#)

Declared In

SCSArticleQuery.h

searchTerm

Specifies the search term to be used to query articles. This property cannot be used with [articleId](#), [sortOrder](#), or [sortByField](#).

Declaration

```
@property (nullable, nonatomic, copy, readwrite) NSString *searchTerm
```

Declared In

SCSArticleQuery.h

pageSize

Specifies the number of articles to fetch.

Declaration

```
@property (nonatomic, assign, readwrite) NSUInteger pageSize
```

Discussion

An org does not return more than 100 results per page.

Declared In

SCSArticleQuery.h

sortOrder

If the [categories](#) property is populated, this property specifies the sort order to be used.

Declaration

```
@property (nonatomic, assign, readwrite) SCArticleSortOrder sortOrder
```

See Also

- [SCArticleSortOrder](#)

Declared In

SCArticleQuery.h

sortByField

If the [categories](#) property is populated, this property specifies the article field used for sorting the article list.

Declaration

```
@property (nonatomic, assign, readwrite) SCArticleSortByField sortByField
```

See Also

- [SCArticleSortByField](#)

Declared In

SCArticleQuery.h

KnowledgeFramework String Constants

List of string constants associated with the Service SDK Knowledge framework.

Table 10: String Constants

String Token	Description	Default Value
ServiceCloud.DeveloperUtility.VersionLabel	Development status message	v%@
ServiceCloud.DeveloperUtility.Title	Development title label	Service Cloud SDK
ServiceCloud.DeveloperUtility.Network.Title	Development network section title	Networking
ServiceCloud.DeveloperUtility.Data.Title	Development Data section title	Data
ServiceCloud.DeveloperUtility.Network.Prompt	Development network section title	Changes take effect after reopening the service UI
ServiceCloud.DeveloperUtility.Network.MockData	Development network mock data title	Use mock network data
ServiceCloud.DeveloperUtility.Data.ClearCache	Development data clear cache title	Clear Cache
ServiceCloud.DeveloperUtility.Data.ClearCache.AlertMessage	Development clear cache alert message	Are you sure you want to clear the cache?
ServiceCloud.DeveloperUtility.Data.CancelButtonTitle	Development clear cache alert cancel button title	Cancel
ServiceCloud.DeveloperUtility.Data.OKButtonTitle	Development clear cache alert ok button title	OK
ServiceCloud.ServiceInterface.ClearCache.AlertMessage	Alert message when clearcache is set and help is opened	Please restart the app to clear out cache

String Token	Description	Default Value
ServiceCloud.ServiceInterface.ClearCacheAlertOKButtonTitle	Development clear cache alert ok button title	OK
ServiceCloud.SupportHome.EmptyArticleListMsgTitle	Main message displayed when the articlelist is empty in support home	Well, this is embarrassing
ServiceCloud.SupportHome.EmptyArticleListMsgDescription	Second line of the message displayed when article list is empty in Support home	There are no articles in this category
ServiceCloud.SupportHome.ExpandCategory	Accessibility message to expand category	Expand category %@
ServiceCloud.SupportHome.CollapseCategory	Accessibility message to collapse category	Collapse category %@
ServiceCloud.SupportHome.HelpTitle	Title of a support home controller	HELP
ServiceCloud.SupportHome.Close	Accessibility message of the close button	Close
ServiceCloud.SupportHome.InvalidDataCategoryOrGroupErrMsgTitle	Error message title displayed for invalid data category group	Invalid data category group or data category selected.
ServiceCloud.SupportHome.InvalidDataCategoryOrGroupErrMsgDesc	Error message description displayed for invalid data category	Please contact the administrator.
ServiceCloud.ArticleDetail.Back	Accessibility message of the back button on Article Detail	Back
ServiceCloud.ArticleDetail.HelpTitle	Title of a article detail controller	HELP
ServiceCloud.ArticleDetail.EmptyMsg	Message to display when no article has yet been selected	Select an article or category to explore solutions
ServiceCloud.ArticleDetail.MinimizeButton	Accessibility Message of the minimize button on Article Detail	Minimize
ServiceCloud.DataCategoryDetail.NoDataMsgTitle	Title to display when Data Category Detail View is empty	This category is empty
ServiceCloud.DataCategoryDetail.NoDataMsgDescription	Description to display when Data Category Detail View is empty	There are no articles in this category
ServiceCloud.DataCategoryDetail.HelpTitle	Title of a data category detail controller	HELP
ServiceCloud.DataCategoryDetail.SubcategoryHeader	Title of the Sub category header	Sub Categories
ServiceCloud.OverlayPresentation.Close	Accessibility label for the overlay controller	Close
ServiceCloud.ShowMoreCellView.ShowMoreLabelText	Label for ShowMoreCell title	Show More
ServiceCloud.ShowMoreCellView.ErrorMessage	Label for ShowMoreCell title	Unable to load more articles...
ServiceCloud.DataCategoryDetail.NoSearchResultMsgTitle	Title to display when no search results are found	No results found
ServiceCloud.DataCategoryDetail.NoSearchResultMsgDescription	Description to display when no search results are found	Please make sure your words are spelled correctly, or use different keywords.
ServiceCloud.EmptySearchView.TitleMessage	Empty search page title string	Help Search

String Token	Description	Default Value
ServiceCloud.EmptySearchView.DescriptionMessage	Empty search page description string	Search our help articles and solutions

Case Management Reference Docs

Reference documentation for the Case Management component of the Service SDK.

[SCCaseInterface Class Reference](#)

Primary access point when interacting with the case management interface.

[SCSCaseListViewController Class Reference](#)

View controller that shows a list of cases for a particular user.

[SCSCaseListViewControllerDelegate Protocol Reference](#)

Delegate protocol used to receive events about the actions performed by the case list view controller.

[SCSCasePublisherResult Constants Reference](#)

Possible result types for the case publisher view controller.

[SCSCasePublisherViewController Class Reference](#)

View controller that lets users create a case and submit it to Service Cloud.

[SCSCasePublisherViewControllerDelegate Protocol Reference](#)

Delegate protocol used to receive events about the actions performed by the case publisher view controller.

[SCSCaseDetailViewController Class Reference](#)

View controller that lets users see the details for a particular case.

[SCSCaseDetailViewControllerDelegate Protocol Reference](#)

Delegate protocol used to receive events about the actions performed by the case detail view controller.

[SCSCaseNotification Class Reference](#)

Concrete subclass of SCNotification representing a push notification for a case.

[CasesFramework String Constants](#)

List of string constants associated with the Service SDK Cases framework.

SCCaseInterface Class Reference

Primary access point when interacting with the case management interface.

Get an instance of this class with the [cases](#) property on the [SCServiceCloud](#) shared instance.

caseListName

The Salesforce Unique name of the case list to be displayed.

Declaration

```
@property (nonatomic, copy) NSString *caseListName
```

Declared In

```
SCCaseInterface.h
```


caseCreateActionName

The name of the action used to create a case.

Declaration

```
@property (nonatomic, copy) NSString *caseCreateActionName
```

Declared In

```
SCCaseInterface.h
```

enabled

Indicates whether or not the Cases portion of the Service Cloud SDK is enabled.

Declaration

```
@property (nonatomic, assign, getter=isEnabled) BOOL enabled
```

Declared In

```
SCCaseInterface.h
```

caseMetadataCacheRetentionTimeInterval

The duration (specified in seconds) after which the case metadata is deleted. The default value is 24 hours.

Declaration

```
@property (nonatomic, assign) NSTimeInterval caseMetadataCacheRetentionTimeInterval
```

Declared In

```
SCCaseInterface.h
```

remoteLoggingEnabled

Determines whether session logs are sent for collection. Logs sent remotely do not collect personal information. Unique IDs are created for tying logs to sessions, and those IDs cannot be correlated back to specific users.

Declaration

```
@property (nonatomic) BOOL remoteLoggingEnabled
```

Discussion

Default: YES

Declared In

```
SCCaseInterface.h
```

CATEGORY: CaseUI Methods

– setInterfaceVisible:animated:completion:

Controls the visibility of the case publisher interface, with an optional animation.

Declaration

```
- (void)setInterfaceVisible:(BOOL)interfaceVisible animated:(BOOL)animated
completion:(dispatch_block_t)completionBlock
```

Parameters

Name	Description
interfaceVisible	YES to make the appearance visible, otherwise it will be hidden.
animated	YES to animate, otherwise NO.
completionBlock	Optional block to invoke when the interface change is completed.

Declared In

SCCaseInterface+CaseUI.h

interfaceVisible

Indicates whether or not the case publisher interface is currently visible. Setting this value will present or dismiss the interface in an unanimated fashion.

Declaration

```
@property (nonatomic, assign, getter=isInterfaceVisible) BOOL interfaceVisible
```

See Also

- - [setInterfaceVisible:animated:completion:](#)

Declared In

SCCaseInterface+CaseUI.h

SCSCaseListViewController Class Reference

View controller that shows a list of cases for a particular user.

This controller assumes that it will be presented within an instance of `UINavigationController`. If you aren't pushing this controller onto a navigation controller stack, present it within a new navigation controller instance.

```
SCSCaseListViewController *caseController = [SCSCaseListViewController new];
UINavigationController *navigationController = [[UINavigationController alloc]
initWithRootViewController:caseController];
[self presentViewController:navigationController animated:YES completion:nil];
```

Warning: If you instantiate this view controller and display it manually, you'll also need to display [SCSCaseDetailViewController](#). When a user selects a case from the case list, present your Case Detail view controller from the `caseList:selectedCaseWithId:` method in your [SCSCaseListViewControllerDelegate](#) implementation. If you don't do this, nothing will happen when a user taps on a specific case in the case list!

delegate

Delegate for the case list view controller.

Declaration

```
@property (nonatomic, weak, nullable) NSObject<SCSCaseListViewControllerDelegate>
*delegate
```

See Also

- [SCSCaseListViewControllerDelegate](#)

Declared In

SCSCaseListViewController.h

SCSCaseListViewControllerDelegate Protocol Reference

Delegate protocol used to receive events about the actions performed by the case list view controller.

See [SCSCaseListViewController](#) for more information.

– caseListDidLoad:

Tells the delegate that the case list loaded.

Declaration

```
- (void)caseListDidLoad:(SCSCaseListViewController *)caseList
```

Parameters

Name	Description
caseList	Case list controller sending the message.

See Also

- [SCSCaseListViewController](#)

Declared In

SCSCaseListViewController.h

– caseList:didReceiveError:

Tells the delegate when there's an error loading the case list.

Declaration

```
- (void)caseList:(SCSCaseListViewController *)caseList didReceiveError:(NSError *)error
```

Parameters

Name	Description
error	Error associated with loading the case list view controller.

See Also

- [SCSCaseListViewController](#)

Declared In

SCSCaseListViewController.h

– caseList:selectedCaseWithId:

Tells the delegate when a case is selected from the case list. If you've manually displayed a [SCSCaseListViewController](#), then use this method to display [SCSCaseDetailViewController](#) with the `caseId` parameter. If you didn't manually display a [SCSCaseListViewController](#), then the Case Detail view appears automatically.

Declaration

```
- (void)caseList:(SCSCaseListViewController *)caseList selectedCaseWithId:(NSString *)caseId
```

Parameters

Name	Description
caseList	Instance of the <code>CaseListViewController</code> class currently presented.
caseId	ID of the selected case.

See Also

- [SCSCaseListViewController](#)

Declared In

SCSCaseListViewController.h

SCSCasePublisherResult Constants Reference

Possible result types for the case publisher view controller.

This value is used by the [SCSCasePublisherViewControllerDelegate](#) protocol method.

Definition

```
typedef NS_ENUM(NSInteger, SCSCasePublisherResult ) {

    SCSCasePublisherResultCancelled = 0,

    SCSCasePublisherResultCaseCreated,

    SCSCasePublisherResultError,

};
```

Constants

SCSCasePublisherResultCancelled

The user canceled the case publisher.

SCSCasePublisherResultCaseCreated

A case was successfully created.

SCSCasePublisherResultError

An error was received while creating a case.

Declared In

SCSCasePublisherViewController.h

SCSCasePublisherViewController Class Reference

View controller that lets users create a case and submit it to Service Cloud.

This controller assumes that it will be presented within an instance of `UINavigationController`. If you aren't pushing this controller onto a navigation controller stack, present it within a new navigation controller instance.

```
SCSCasePublisherViewController *caseController = [SCSCasePublisherViewController new];
UINavigationController *navigationController = [[UINavigationController alloc]
    initWithRootViewController:caseController];
[self presentViewController:navigationController animated:YES completion:nil];
```

delegate

Delegate for the case publisher view controller.

Declaration

```
@property (nonatomic, weak, nullable) NSObject<SCSCasePublisherViewControllerDelegate>
    *delegate
```

See Also

- [SCSCasePublisherViewControllerDelegate](#)

Declared In

SCSCasePublisherViewController.h

SCSCasePublisherViewControllerDelegate Protocol Reference

Delegate protocol used to receive events about the actions performed by the case publisher view controller.

See [SCSCasePublisherViewController](#) for more information.

– casePublisher:didSubmitWithResult:withCaseId:error:

Tells the delegate when the case is submitted. If the submission failed, this method passes the error to the delegate.

Declaration

```
- (void)casePublisher:(SCSCasePublisherViewController *)publisher
didSubmitWithResult:(SCSCasePublisherResult)result withCaseId:(nullable NSString *)caseId
error:(nullable NSError *)error
```

Parameters

Name	Description
<i>publisher</i>	Case publisher controller sending the message.
<i>result</i>	Result of the user's action.
<i>caseId</i>	ID of the created case if one was created, and if the ID can be determined.
<i>error</i>	Error returned, if any.

See Also

- [SCSCasePublisherViewController](#)
- [SCSCasePublisherResult](#)

Declared In

SCSCasePublisherViewController.h

– casePublisher:fieldsToHideFromCaseFields:

Allows the delegate to hide some of the case fields from showing in the Case Publisher.

Declaration

```
- (NSSet<NSString*> *)casePublisher:(SCSCasePublisherViewController *)publisher
fieldsToHideFromCaseFields:(NSArray<NSString*> *)availableFields
```

Parameters

Name	Description
<i>publisher</i>	Case publisher controller sending the message.
<i>availableFields</i>	List of fields in the Case Publisher layout.

Return Value

Set of fields to be hidden in the Case Publisher.

Declared In

SCSCasePublisherViewController.h

– casePublisher:valuesForHiddenFields:

Allows the delegate to pass in the case field values while submitting the case.

Declaration

```
- (NSDictionary<NSString*,id> *)casePublisher:(SCSCasePublisherViewController *)publisher
  valuesForHiddenFields:(NSSet<NSString*> *)hiddenFields
```

Parameters

Name	Description
<i>publisher</i>	Case publisher controller sending the message.
<i>hiddenFields</i>	Set of hidden fields in the Case Publisher.

Return Value

NSDictionary in which each entry maps the field name to the field value to be used during case submission.

Declared In

SCSCasePublisherViewController.h

– casePublisher:viewForResult:withCaseId:error:

Allows the delegate to supply a custom view to display to the user when the provided result is reached.

Declaration

```
- (nullable UIView *)casePublisher:(SCSCasePublisherViewController *)publisher
  viewForResult:(SCSCasePublisherResult)result withCaseId:(nullable NSString *)caseId
  error:(nullable NSError *)error
```

Parameters

Name	Description
<i>publisher</i>	Case publisher controller sending the message.
<i>result</i>	Result of the case publisher.
<i>caseId</i>	ID of the created case if one was created, and if the ID can be determined.
<i>error</i>	Error returned, if any.

Return Value

UIView instance configured for showing the result to the user, or `nil` if the default should be used.

Discussion

If this method is not implemented, or the result is `nil`, the case publisher will automatically provide a default value. The returned view will be sized as needed to fit the available space on the case publisher.

Declared In

SCSCasePublisherViewController.h

– `shouldEnableCaseDeflectionForPublisher:`

Allows the delegate to disable the case deflection feature in the Case Publisher. By default the case deflection is enabled.

Declaration

```
- (BOOL)shouldEnableCaseDeflectionForPublisher:(SCSCasePublisherViewController *)publisher
```

Parameters

Name	Description
<code>publisher</code>	Case publisher controller sending the message.

Return Value

YES if the case publisher should display case deflection articles.

Declared In

`SCSCasePublisherViewController.h`

– `casePublisher:fieldsForCaseDeflection:`

Allows the delegate to specify the case fields for which case deflection articles are displayed.

Declaration

```
- (NSSet<NSString*> *)casePublisher:(SCSCasePublisherViewController *)publisher  
fieldsForCaseDeflection:(NSArray<NSString*> *)availableFields
```

Parameters

Name	Description
<code>publisher</code>	Case publisher controller sending the message.
<code>availableFields</code>	List of editable text or textarea fields in the Case Publisher layout.

Return Value

Set of fields to be used for forming case deflection searchterm.

Declared In

`SCSCasePublisherViewController.h`

SCSCaseDetailViewController Class Reference

View controller that lets users see the details for a particular case.

Initialize this view controller with a case ID using `initWithCaseId:`.

– `initWithCaseId:`

Initializes the `SCSCaseDetailViewController` with a case ID.

Declaration

```
- (instancetype)initWithCaseId:(NSString *)caseId
```

Parameters

Name	Description
caseId	ID of the case to be displayed.

Declared In

SCSCaseDetailViewController.h

delegate

Delegate for the case detail view controller.

Declaration

```
@property (nonatomic, weak, nullable) NSObject<SCSCaseDetailViewControllerDelegate> *delegate
```

See Also

- [SCSCaseDetailViewControllerDelegate](#)

Declared In

SCSCaseDetailViewController.h

SCSCaseDetailViewControllerDelegate Protocol Reference

Delegate protocol used to receive events about the actions performed by the case detail view controller.

See [SCSCaseDetailViewController](#) for more information.

– caseDetail:fieldsToHideFromCaseFields:

Allows the delegate to hide some of the case fields from showing in the Case Detail.

Declaration

```
- (NSSet<NSString*> *)caseDetail:(SCSCaseDetailViewController *)caseDetailController fieldsToHideFromCaseFields:(NSArray<NSString*> *)availableFields
```

Parameters

Name	Description
caseDetailController	Case Detail View controller sending the message.
availableFields	List of fields available for display in Case Detail. The parameter is of type NSArray<NSString*>.

Return Value

Set of fields to be hidden in the Case Detail. Return expected is of type `NSSet<NSString*>*`.

Declared In

`SCSCaseDetailViewController.h`

SCSCaseNotification Class Reference

Concrete subclass of `SCSNotification` representing a push notification for a case.

caseId

The case ID for the notification.

Declaration

```
@property (nonatomic, strong, readonly) NSString *caseId
```

Declared In

`SCSNotification.h`

CasesFramework String Constants

List of string constants associated with the Service SDK Cases framework.

Table 11: String Constants

String Token	Description	Default Value
<code>ServiceCloud.CasePublisher.Title</code>	Title of the Case Publisher controller	CONTACT
<code>ServiceCloud.CasePublisher.Close</code>	Accessibility label for the close button on Case Publisher	Close
<code>ServiceCloud.CasePublisher.SubmitButton</code>	Label of the submit button on Case Publisher	Send
<code>ServiceCloud.CasePublisher.CasePublisherCloseAlertTitle</code>	Title of the alert shown on closing case publisher	Save Your Message?
<code>ServiceCloud.CasePublisher.SaveDiscardAlertSaveAction</code>	Label of the save action on alert shown when closing case publisher	Save
<code>ServiceCloud.CasePublisher.SaveDiscardAlertDiscardAction</code>	Label of the discard action on alert shown when closing case publisher	Discard
<code>ServiceCloud.CasePublisher.DisabledSubmitButton</code>	Accessibility Label of the disabled submit button on Case Publisher	SUBMIT Dimmed
<code>ServiceCloud.CasePublisher.RequiredField</code>	Accessibility Label of the required field on Case Publisher	Required
<code>ServiceCloud.CasePublisher.NextFieldButton</code>	Accessibility label for the next button on Case Publisher	Next field

String Token	Description	Default Value
ServiceCloud.CasePublisher.PreviousFieldButton	Accessibility label for the previous button on Case Publisher	Previous field
ServiceCloud.CasePublisher.PicklistAccessibilityLabelFormat	Accessibility label format for the picklist field. Fieldname, value	%@, %@
ServiceCloud.CasePublisher.SubmitInProgress	Label of the submit in progress on Case Publisher	SUBMITTING CASE...
ServiceCloud.CasePublisher.SubmissionComplete	Label of the submit button when case is created.	CASE CREATED
ServiceCloud.CasePublisher.SuccessMessage	Default text when case is created on Case Publisher	Your message has been sent to our support team.\n\nWe will respond as soon as we can.\n\nThank you.
ServiceCloud.CasePublisher.InvalidEmailMessage	Case Publisher Invalid Email Address Message	Invalid Email
ServiceCloud.CasePublisher.SelectAValue	Case Publisher Select a value picklist string	Select a value...
ServiceCloud.CasePublisher.DeflectionLoadingLabel	Case Deflection Loading string	Looking for Relevant Articles...
ServiceCloud.CasePublisher.MinimizedDeflectionResult	Case Deflection result string	%@ Relevant Articles Found
ServiceCloud.CasePublisher.MinimizedDeflectionMoreResult	Case Deflection result string when more than one page of articles	%@+ Relevant Articles Found
ServiceCloud.CaseList.Title	Case List Title string	MY CASES
ServiceCloud.CaseList.CaseInboxEmpty	Case List empty list message	Your case inbox is empty
ServiceCloud.CaseList.ContactSupport	Case List contact support button title	Contact Support
ServiceCloud.CaseList.Close	Accessibility label for the close button on Case List	Close
ServiceCloud.CaseList.Create	Accessibility label for the create button on Case List	Create
ServiceCloud.CaseList.Unread	Accessibility label for the unread image on Case List	Unread
ServiceCloud.CaseList.CellAccessibilityFormatForUnreadCase	Accessibility format for the cell on Case List corresponds to unread, CaseCreatedDate, CaseSubject	%@, %@, %@
ServiceCloud.CaseList.CellAccessibilityFormatForReadCase	Accessibility format for the cell on Case List corresponds to CaseCreatedDate, CaseSubject	%@, %@
ServiceCloud.CaseDetail.NoMessageToDisplay	Case Detail empty feed message	No message to display
ServiceCloud.CaseDetail.Today	Case Detail / List DateTime string for yesterday	Today %@

String Token	Description	Default Value
ServiceCloud.CaseDetail.Yesterday	Case Detail / List DateTime string for today	Yesterday %@
ServiceCloud.CaseDetail.MinuteAgo	Case Detail / List 1 Minute ago	1 minute ago
ServiceCloud.CaseDetail.MinutesAgo	Case Detail / List Minutes ago	%lu minutes ago
ServiceCloud.CaseDetail.JustNow	Case Detail Just Now	Just Now
ServiceCloud.CaseDetail.InputPlaceholder	Case Feed Input placeholder	Type a message
ServiceCloud.CaseDetail.Subtitle	Case Detail / Subtitle Format For Case Number	CASE #%@
ServiceCloud.CaseDetail.SendButtonTitle	Case Detail Send Button Title	Send
ServiceCloud.CaseDetail.Close	Accessibility label for the close button on Case Detail	Close

Live Agent Chat Reference Docs

Reference documentation for the Live Agent Chat component of the Service SDK.

[SCSChat Class Reference](#)

The SCSChat class is the main interface to the Live Agent Chat SDK.

[SCSChatConfiguration Class Reference](#)

A SCSChatConfiguration object contains configuration information for a Live Agent Chat session.

[SCSChatDelegate Protocol Reference](#)

The SCSChatDelegate protocol provides information about the Live Agent Chat session.

[SCSChatEndReason Constants Reference](#)

Reasons why a Live Agent Session may have ended.

[SCSChatErrorCode Constants Reference](#)

Live Agent Chat ErrorCode definitions.

[SCSChatSessionState Constants Reference](#)

Full list of Session states the Live Agent Chat framework can exhibit.

[SCSPrechatObject Class Reference](#)

A SCSPrechatObject specifies a pre-chat field that you can send directly to the agent. This object contains a label and a value.

[SCSPrechatPickerObject Class Reference](#)

An SCSPrechatPickerObject specifies a pre-chat picker field that is displayed before a chat session is initiated.

[SCSPrechatPickerOption Class Reference](#)

An SCSPrechatPickerOption specifies an option inside a pre-chat picker field that is displayed before a chat session is initiated.

[SCSPrechatTextInputObject Class Reference](#)

An SCSPrechatTextInputObject specifies a pre-chat text input field that is displayed before a chat session is initiated.

[ChatFramework String Constants](#)

List of string constants associated with the Service SDK Live Agent Chat framework.

SCSChat Class Reference

The SCSChat class is the main interface to the Live Agent Chat SDK.

This object manages the flow of Chat sessions throughout the lifetime of the app.

SCSChat conforms to a multicast delegate model for messaging. Any class which implements the [SCSChatDelegate](#) protocol can be added to a list of delegates to receive messages asynchronously.

CATEGORY: Properties

configuration

A reference to the [SCSChatConfiguration](#) object provided to the session on start.

Declaration

```
@property (nonatomic, readonly, strong) SCSChatConfiguration *configuration
```

Declared In

SCSChat.h

CATEGORY: Session Control

– startSessionWithConfiguration:

This method will begin the process for starting a Live Agent Chat session.

Declaration

```
– (void)startSessionWithConfiguration:(SCSChatConfiguration *)config
```

Parameters

Name	Description
config	The SCSChatConfiguration object which represents the session configuration.

Discussion

Equivalent to invoking [\[SCSChat startSessionWithConfiguration:completion:\]](#) and providing a `nil` block.

See Also

- [\[SCSChat startSessionWithConfiguration:completion:\]](#)

Declared In

SCSChat.h

– startSessionWithConfiguration:completion:

This method will begin the process for starting a Live Agent Chat session.

Declaration

```
- (void)startSessionWithConfiguration:(SCSChatConfiguration *)config
completion:(SCSChatCompletionHandler)block
```

Parameters

Name	Description
config	The SCSChatConfiguration object which represents the session configuration.
block	Completion block which will be executed when the session has been fully connected to all services. NOTE: when the block is executed the session is active and waiting for an agent to join. NOTE: the NSError returned in the block will be <code>nil</code> on success.

Discussion

NOTE: Calling [\[SCSChat stopSession\]](#) at any point after calling [\[SCSChat startSessionWithConfiguration:\]](#) will not necessarily terminate a session in progress. Once the user has moved past the pre chat phase this will trigger a prompt asking the user if they wish to terminate the session.

Declared In

SCSChat.h

– stopSession

Stops an active or connecting session.

Declaration

```
- (void)stopSession
```

Discussion

If the user has not moved past the pre chat phase this will immediately terminate the session and clean up all resources.

Equivalent to invoking [\[SCSChat stopSessionWithCompletion:\]](#) and providing a `nil` block.

See Also

- [\[SCSChat stopSessionWithCompletion:\]](#)

Declared In

SCSChat.h

– stopSessionWithCompletion:

Stops an active or connecting session.

Declaration

```
- (void)stopSessionWithCompletion:(SCSChatCompletionHandler)block
```

Parameters

Name	Description
block	Completion block which will be executed when the session has fully stopped, and all connected services have been torn down. NOTE: the NSError returned in the block will be nil on success.

Discussion

If the user has not moved past the pre chat phase this will immediately terminate the session and clean up all resources.

Declared In

SCSChat.h

– determineAvailabilityWithConfiguration:completion:

Queries Live Agent to determine agent availability to accept a chat session.

Declaration

```
- (void)determineAvailabilityWithConfiguration:(SCSChatConfiguration *)config
completion:(SCSChatAvailabilityHandler)block
```

Parameters

Name	Description
config	The SCSChatConfiguration object which represents the session configuration.
block	The completion block that executes when the availability response returns.

Declared In

SCSChat.h

CATEGORY: Delegate Management**– addDelegate:**

Adds an instance of an NSObject implementing the [SCSChatDelegate](#) protocol to the list of delegates to notify.

Declaration

```
- (void)addDelegate:(NSObject<SCSChatDelegate> *)delegate
```

Parameters

Name	Description
delegate	NSObject instance to add.

Declared In`SCSChat.h`**– removeDelegate:**

Removes an instance of an `NSObject` implementing the [SCSChatDelegate](#) protocol from the list of delegates to notify.

Declaration

```
- (void)removeDelegate:(NSObject<SCSChatDelegate> *)delegate
```

Parameters

Name	Description
delegate	<code>NSObject</code> instance to remove.

Declared In`SCSChat.h`

SCSChatConfiguration Class Reference

A `SCSChatConfiguration` object contains configuration information for a Live Agent Chat session.

CATEGORY: Initialization**– initWithLiveAgentPod:orgId:deploymentId:buttonId:**

Instantiates an `SCSChatConfiguration` object for use with [\[SCSChat startSessionWithConfiguration:\]](#)

Declaration

```
- (instancetype)initWithLiveAgentPod:(NSString *)liveAgentPod orgId:(NSString *)orgId  
deploymentId:(NSString *)deploymentId buttonId:(NSString *)buttonId
```

Parameters

Name	Description
liveAgentPod	The hostname for the LiveAgent endpoints that your organization has been assigned.
orgId	The Salesforce 15 character Organization ID.
deploymentId	The unique ID for the deployment that this client will be configured to use.
buttonId	The unique ID for the chat configuration that this client will use.

Return Value

The `SCSChatConfiguration` instance.

Declared In`SCSChatConfiguration.h`**CATEGORY: Deployment Configuration****liveAgentPod**

The hostname for the LiveAgent endpoints that your organization has been assigned.

Declaration

```
@property (nonatomic, strong, readonly) NSString *liveAgentPod
```

Declared In`SCSChatConfiguration.h`**organizationId**

The Salesforce 15 character Organization ID.

Declaration

```
@property (nonatomic, strong, readonly) NSString *organizationId
```

Declared In`SCSChatConfiguration.h`**deploymentId**

The unique ID for the deployment that this client will be configured to use.

Declaration

```
@property (nonatomic, strong, readonly) NSString *deploymentId
```

Declared In`SCSChatConfiguration.h`**buttonId**

The unique ID for the chat configuration that this client will use.

Declaration

```
@property (nonatomic, strong, readonly) NSString *buttonId
```

Declared In`SCSChatConfiguration.h`

CATEGORY: Session Behavior

prechatFields

An array of [SCSPrechatObject](#) objects defining the custom information this session will provide.

Declaration

```
@property (nonatomic, readonly, strong) NSMutableArray<SCSPrechatObject*> *prechatFields
```

Declared In

SCSChatConfiguration.h

presentationStyle

Defines how the Live Agent Chat session is presented when it starts.

Declaration

```
@property (nonatomic, assign) SCSChatPresentationStyle presentationStyle
```

Discussion

Defaults to `SCSChatPresentationStyleNonBlocking`.

Declared In

SCSChatConfiguration.h

remoteLoggingEnabled

Determines whether session logs are sent for collection. Logs sent remotely do not collect personal information. Unique IDs are created for tying logs to sessions, and those IDs cannot be correlated back to specific users.

Declaration

```
@property (nonatomic) BOOL remoteLoggingEnabled
```

Discussion

Default: YES

Declared In

SCSChatConfiguration.h

SCSChatDelegate Protocol Reference

The `SCSChatDelegate` protocol provides information about the Live Agent Chat session.

See [SCSChat](#) for more information about Chat session management.

– chat:stateDidChange:previous:

Delegate method invoked when the Live Agent Chat session state changes.

Declaration

```
- (void) chat:(SCSChat *) chat stateDidChange:(SCSChatSessionState) current
previous:(SCSChatSessionState) state
```

Parameters

Name	Description
chat	SCSChat instance which invoked the method.
current	The new SCSChatSessionState which has been set on the for the Live Agent Chat session.
previous	The previous SCSChatSessionState .

Declared In

SCSChatDelegate.h

– chat:didEndWithReason:error:

Delegate method invoked when a Live Agent session ends.

Declaration

```
- (void) chat:(SCSChat *) chat didEndWithReason:(SCSChatEndReason) reason error:(NSError *) error
```

Parameters

Name	Description
chat	SCSChat instance which invoked the method.
reason	SCSChatEndReason describing why the session has ended.
error	NSError instance describing the error. Error codes can be referenced from SCSChatErrorCode

Declared In

SCSChatDelegate.h

– chat:didError:

Delegate method invoked if an error is raised during a Live Agent Chat session.

Declaration

```
- (void) chat:(SCSChat *) chat didError:(NSError *) error
```

Parameters

Name	Description
chat	SCSChat instance which invoked the error.
error	NSError instance describing the error. Error codes can be referenced from SCSChatErrorCode

Declared In

SCSChatDelegate.h

SCSChatEndReason Constants Reference

Reasons why a Live Agent Session may have ended.

Definition

```
typedef NS_ENUM(NSInteger, SCSChatEndReason ) {  
  
    SCSChatEndReasonUser = 1,  
  
    SCSChatEndReasonAgent = 2,  
  
    SCSChatEndReasonTimeout = 3,  
  
    SCSChatEndReasonSessionError = 4,  
  
};
```

Constants

SCSChatEndReasonUser

User disconnected the session.

SCSChatEndReasonAgent

Agent disconnected the session.

SCSChatEndReasonTimeout

Session ended due to timeout

SCSChatEndReasonSessionError

Session ended due to an error

See Also

- [\[SCSChatDelegate chat:didEndWithReason:error:\]](#)

Declared In

SCSChatEndReason.h

SCSChatErrorCode Constants Reference

Live Agent Chat ErrorCode definitions.

Errors fall into one of several groups (or series) of errors.

If there is an error thrown by a library dependency it will be included as metadata in the userInfo of the error which is returned to the application code.

Definition

```
typedef NS_ENUM(NSInteger, SCSChatErrorCode) {  
  
    SCSChatGenericError = 1000,  
  
    SCSChatInvalidConfiguration = 1001,  
  
    SCSChatExistingSessionError = 1002,  
  
    SCSChatCommunicationError = 2000,  
  
    SCSChatNoAgentsAvailableError = 2001,  
  
    SCSChatSessionCreationError = 2002,  
  
    SCSChatSessionEventError = 2003,  
  
};
```

Constants

SCSChatGenericError

Unclassified error. This results from an unknown or unexpected error state.

SCSChatInvalidConfiguration

The [SCSChatConfiguration](#) object provided to the [SCSChat](#) is invalid.

SCSChatExistingSessionError

Another session is already in progress

SCSChatCommunicationError

Standard communication error. This can be returned from operations dependent on communication between Live Agent or any other remote system.

SCSChatNoAgentsAvailableError

Returned when attempting to start a session when no agents are available to accept the request.

SCSChatSessionCreationError

Returned when attempting to start a session and a network error occurs.

SCSChatSessionEventError

Returned when attempting to perform an action in an established session and a network error occurs.

See Also

- [\[SCSChatDelegate chat:didError:\]](#)
- [\[SCSChatDelegate chat:didEndWithReason:error:\]](#)

Declared In

SCSChatErrorCode.h

SCSChatSessionState Constants Reference

Full list of Session states the Live Agent Chat framework can exhibit.

Definition

```
typedef NS_ENUM(NSInteger, SCSChatSessionState ) {  
  
    SCSChatSessionStateInactive = 0,  
  
    SCSChatSessionStateLoading,  
  
    SCSChatSessionStatePrechat,  
  
    SCSChatSessionStateConnecting,  
  
    SCSChatSessionStateQueued,  
  
    SCSChatSessionStateConnected,  
  
    SCSChatSessionStateEnding,  
  
    SCSChatSessionStateEnded,  
  
};
```

Constants**SCSChatSessionStateInactive**

No active session. There will be no outgoing/incoming Chat traffic.

SCSChatSessionStateLoading

Session is being loaded to begin connection process.

SCSChatSessionStatePrechat

Prechat details are being filled out by the end-user.

SCSChatSessionStateConnecting

A connection with Live Agent servers is being established.

SCSChatSessionStateQueued

A connection has been established, but queueing for next available agent.

SCSChatSessionStateConnected

Connected with an agent to facilitate a chat session.

SCSChatSessionStateEnding

Session is in the process of cleaning up network connections and ending.

SCSChatSessionStateEnded

Session has ended. Will proceed to the inactive state.

See Also

- [\[SCSChatDelegate chat:stateDidChange:previous:\]](#)

Declared In

SCSChatSessionState.h

SCSPrechatObject Class Reference

A SCSPrechatObject specifies a pre-chat field that you can send directly to the agent. This object contains a label and a value.

This object does not prompt the user for information. To create a field that the user can fill in, refer to [SCSPrechatTextInputObject](#).

This object must be added to your chat configuration using [\[SCSChatConfiguration prechatFields\]](#).

CATEGORY: Properties

label

Name of pre-chat detail shown to agent.

Declaration

```
@property (nonatomic, strong, readonly) NSString *label
```

Declared In

SCSPrechatObject.h

value

Value of the pre-chat detail.

Declaration

```
@property (nonatomic, strong, readonly) NSString *value
```

Declared In`SCSPrechatObject.h`**CATEGORY: Initialization****– initWithLabel:value:**

Instantiates an `SCSPrechatObject` object for use with [\[SCSChatConfiguration prechatFields\]](#)

Declaration

```
- (instancetype)initWithLabel:(NSString *)label value:(NSString *)value
```

Parameters

Name	Description
label	The identifying label to show to the agent in the Service Console.
value	The value of the field.

Return Value

The `SCSPrechatObject` instance.

Declared In`SCSPrechatObject.h`

SCSPrechatPickerObject Class Reference

An `SCSPrechatPickerObject` specifies a pre-chat picker field that is displayed before a chat session is initiated.

Use the [required](#) property to specify whether this field must have an option selected before initiating a session.

To send data directly to the agent without user input, see [SCSPrechatObject](#).

This object must be added to your chat configuration using [\[SCSChatConfiguration prechatFields\]](#).

CATEGORY: Properties**required**

Determines if the field must have an option before the pre-chat form can be submitted.

Declaration

```
@property (nonatomic, getter=isRequired) BOOL required
```

Declared In`SCSPrechatPickerObject.h`

options

An array of [SCSPrechatPickerOption](#) for the user to select from in the pre-chat form.

Declaration

```
@property (nonatomic, copy, readonly) NSArray<SCSPrechatPickerOption*> *options
```

Declared In

SCSPrechatPickerObject.h

CATEGORY: Initialization

– initWithLabel:options:

Instantiates an `SCSPrechatPickerObject` object for use with [\[SCSChatConfiguration prechatFields\]](#). This object will be displayed in the pre-chat window for the user to select a value.

Declaration

```
- (instancetype)initWithLabel:(NSString *)label options:(NSArray<SCSPrechatPickerOption*> *)options
```

Parameters

Name	Description
label	The identifying label to show to the agent in the Service Console.
options	An array of SCSPrechatPickerOption for the user to select from in the pre-chat form.

Return Value

The `SCSPrechatPickerObject` instance.

Declared In

SCSPrechatPickerObject.h

– initWithLabel:value:

Instantiates an `SCSPrechatObject` object for use with [\[SCSChatConfiguration prechatFields\]](#)

Declaration

```
- (instancetype)initWithLabel:(NSString *)label value:(NSString *)value
```

Parameters

Name	Description
label	The identifying label to show to the agent in the Service Console.
value	The value of the field.

Return Value

The `SCSPrechatObject` instance.

Declared In

`SCSPrechatObject.h`

SCSPrechatPickerOption Class Reference

An `SCSPrechatPickerOption` specifies an option inside a pre-chat picker field that is displayed before a chat session is initiated.

This object must be added to the options array on an [SCSPrechatPickerObject](#).

CATEGORY: Properties

pickerLabel

The label to display inside of the picker to the end user.

Declaration

```
@property (nonatomic, strong, readonly) NSString *pickerLabel
```

Declared In

`SCSPrechatPickerOption.h`

value

The value to show to the agent in the Service Console.

Declaration

```
@property (nonatomic, strong, readonly) NSString *value
```

Declared In

`SCSPrechatPickerOption.h`

CATEGORY: Initialization

– initWithLabel:value:

Instantiates an `SCSPrechatPickerOption` object for use with [SCSPrechatPickerObject](#). This option will be displayed in the pre-chat picker as an option to select.

Declaration

```
- (instancetype)initWithLabel:(NSString *)pickerLabel value:(NSString *)value
```

Parameters

Name	Description
pickerLabel	The label to display inside of the picker to the end user.

Name	Description
value	The value to show to the agent in the Service Console.

Return Value

The `SCSPrechatPickerOption` instance.

Declared In

`SCSPrechatPickerOption.h`

SCSPrechatTextInputObject Class Reference

An `SCSPrechatTextInputObject` specifies a pre-chat text input field that is displayed before a chat session is initiated.

Use the [required](#) property to specify whether this field must be filled in before initiating a session.

To send data directly to the agent without user input, see [SCSPrechatObject](#).

This object must be added to your chat configuration using [\[SCSChatConfiguration prechatFields\]](#).

CATEGORY: Properties

required

Determines if the field must be filled before the pre-chat form can be submitted.

Declaration

```
@property (nonatomic, getter=isRequired) BOOL required
```

Declared In

`SCSPrechatTextInputObject.h`

maxLength

Sets a maximum length for the text entered in the pre-chat field.

Declaration

```
@property (nonatomic) NSUInteger maxLength
```

Declared In

`SCSPrechatTextInputObject.h`

keyboardType

The keyboard type to be used for the field.

Declaration

```
@property (nonatomic) UIKeyboardType keyboardType
```

Declared In

SCSPrechatTextInputObject.h

autocapitalizationType

The autocapitalization behaviour for the field.

Declaration

```
@property (nonatomic) UITextAutocapitalizationType autocapitalizationType
```

Declared In

SCSPrechatTextInputObject.h

autocorrectionType

The autocorrection behaviour for the field.

Declaration

```
@property (nonatomic) UITextAutocorrectionType autocorrectionType
```

Declared In

SCSPrechatTextInputObject.h

CATEGORY: Initialization**– initWithLabel:**

Instantiates an `SCSPrechatTextInputObject` object for use with [\[SCSChatConfiguration prechatFields\]](#). This object will be displayed in the pre-chat window for the user to fill out.

Declaration

```
- (instancetype) initWithLabel: (NSString *) label
```

Parameters

Name	Description
label	The identifying label to show to the agent in the Service Console.

Return Value

The `SCSPrechatTextInputObject` instance.

Declared In

SCSPrechatTextInputObject.h

– initWithLabel:value:

Instantiates an `SCSPrechatObject` object for use with [\[SCSChatConfiguration prechatFields\]](#)

Declaration

```
- (instancetype)initWithLabel:(NSString *)label value:(NSString *)value
```

Parameters

Name	Description
label	The identifying label to show to the agent in the Service Console.
value	The value of the field.

Return Value

The `SCSPrechatObject` instance.

Declared In

`SCSPrechatObject.h`

ChatFramework String Constants

List of string constants associated with the Service SDK Live Agent Chat framework.

Table 12: String Constants

String Token	Description	Default Value
<code>ServiceCloud.Chat.Dialog.General.Title</code>	Title of many alert dialogs and views.	LIVE CHAT
<code>ServiceCloud.Chat.Dialog.General.Cancel</code>	Negative option in generic alert dialogs.	Cancel
<code>ServiceCloud.Chat.Dialog.General.Confirm</code>	Affirmative option in generic alert dialogs.	OK
<code>ServiceCloud.Chat.Dialog.PreChat.Content</code>	String presented to mobile user above a number of text fields that require input before submitting a chat request.	Just a couple things before we connect you with an expert...
<code>ServiceCloud.Chat.Dialog.PreChat.Confirm</code>	Label of button for mobile user to press once they have completed necessary text fields and are ready to chat with an agent.	CHAT WITH AN AGENT
<code>ServiceCloud.Chat.Dialog.EndSession.Title</code>	Title of end session alert dialog presented when a mobile user attempts to end the session.	END SESSION
<code>ServiceCloud.Chat.Dialog.EndSession.Content</code>	Message body in end session alert dialog presented when a mobile user attempts to end the session.	Are you sure you wish to end the chat session?
<code>ServiceCloud.Chat.Dialog.EndSession.Confirm</code>	Affirmative option in end session alert dialog presented when a mobile user attempts to end the session.	End

String Token	Description	Default Value
ServiceCloud.Chat.Dialog.Photo.TakePicture	Option presented to mobile user for sending a file to an agent by taking a new photo with the camera.	Take Photo
ServiceCloud.Chat.Dialog.Photo.Library	Option presented to mobile user for sending a file to an agent by selecting a previously captured image from the device's photo library.	Photo Library
ServiceCloud.Chat.Dialog.NoAgents.Content	Alert message body displayed to mobile user when no agents are available to chat with.	No agents are available to service your request. Please try again later.
ServiceCloud.Chat.Dialog.SessionTerminated.Title	Alert title displayed to mobile user when their chat session has ended.	Session Ended
ServiceCloud.Chat.Dialog.SessionTerminated.Content	Alert message body displayed to mobile user when an agent has ended the chat session.	The agent has ended the session.
ServiceCloud.Chat.Dialog.PreChat.SubmissionError	Label of the submit button on prechat in an error condition	Submission Error
ServiceCloud.Chat.Dialog.PreChat.SubmissionInProgress	Label of the submit button on prechat when submit request is in progress	Submit In Progress
ServiceCloud.Chat.Dialog.PreChat.SubmissionComplete	Label of the submit button on prechat when submit is completed	Submission Complete
ServiceCloud.Chat.Dialog.PreChat.Picklist.SelectAValue	Default picklist option in prechat picklist	Select a value...
ServiceCloud.Chat.Status.Connecting	Status message displayed to mobile user when connecting to chat servers.	Connecting...
ServiceCloud.Chat.Status.Active.Now	Status message displayed to mobile user when agent is currently active in the chat session.	active now
ServiceCloud.Chat.Status.Inactive.SingleMinute	Status message displayed to mobile user when agent has been inactive for a minute.	active 1 min ago
ServiceCloud.Chat.Status.Inactive.MultipleMinutes	Status message displayed to mobile user when agent has been inactive for a specific number of minutes.	active %ld mins ago
ServiceCloud.Chat.Status.Inactive.Hour	Status message displayed to mobile user when agent has been inactive for over an hour.	active over an hour ago
ServiceCloud.Chat.Status.Waiting	Status message displayed to mobile user when waiting for agent to connect to a session.	Waiting for Agent

String Token	Description	Default Value
ServiceCloud.Chat.System.FileTransfer.Requested	Message displayed to mobile user when agent has initiated a file transfer.	FILE TRANSFER REQUESTED
ServiceCloud.Chat.System.FileTransfer.Cancelled	Message displayed to mobile user when agent has cancelled a previously initiated file transfer.	FILE TRANSFER CANCELLED
ServiceCloud.Chat.System.FileTransfer.Complete	Message displayed to mobile user when file transfer to the agent has been successful.	FILE TRANSFER COMPLETE
ServiceCloud.Chat.System.FileTransfer.Failed	Message displayed to mobile user when file transfer to the agent has failed.	FILE TRANSFER FAILED
ServiceCloud.Chat.Input.Placeholder	Placeholder text inside chat input field.	Type a message
ServiceCloud.Chat.Input.Send	Label of button to send a message to the agent.	Send

SOS Reference Docs

Reference documentation for the SOS component of the Service SDK.

[SOSAgentAvailability Class Reference](#)

The SOSAgentAvailability class allows you to configure periodic polling against a single SOS deployment for your organization.

[SOSAgentAvailabilityDelegate Protocol Reference](#)

Delegate protocol for SOSAgentAvailability

[SOSAgentAvailabilityStatusType Constants Reference](#)

Enum for the Availability Status updates generated by the didChange: event

[SOSCameraType Constants Reference](#)

Sets the starting view for the current SOS Session.

[SOSConnectingBaseViewController Class Reference](#)

The SOSConnectingBaseViewController serves as the base controller which manages interactions between the UI and SOS backend for the connecting phase of SOS.

[SOSConnectingViewController Protocol Reference](#)

Protocol which defines properties and methods which are required for all ViewControllers which will accommodate the role for onboarding in SOS.

[SOSDelegate Protocol Reference](#)

The SOSDelegate protocol provides information about the SOS session.

[SOSErrorCode Constants Reference](#)

SOS ErrorCode definitions.

[SOSLogger Class Reference](#)

SOSLogging singleton which manages all logging messages generated by SOS.

[SOSLoggerProtocol Protocol Reference](#)

Allows classes to listen to log messages generated by SOS.

[SOSLogStream Constants Reference](#)

List of Logging types which can be watched.

[SOSLogTimerCode Constants Reference](#)

SOSLog Timer Code enum. See the schema for information:

<https://github.com/goinstant/schema/blob/master/schemas/common/messages/timer.json>

[SOSMaskable Protocol Reference](#)

Protocol for objects that can be registered as maskable. Anything implementing this protocol can be registered with the SOS Masking class, which will send the messages declared below at the appropriate times to enable/disable the masks during an SOS session.

[SOSMaskedTextField Class Reference](#)

Custom UITextField that auto-masks itself when screen sharing is enabled during an SOS session. Otherwise identical to the default UITextField.

[SOSMasking Class Reference](#)

The SOSMasking class is responsible for managing the field masking applied during an SOS session. Sensitive fields or sections of the application may be masked to prevent them from appearing to the customer service agent in the video feed.

[SOSMaskState Constants Reference](#)

Enumeration of the various states a given mask can be in.

[SOSNetworkStatus Constants Reference](#)

Defines the discrete network conditions which can occur within an SOS Session.

[SOSOnboardingBaseViewController Class Reference](#)

The SOSOnboardingBaseViewController serves as the base controller which manages interactions between the UI and SOS backend for the onboarding phase of SOS.

[SOSOnboardingViewController Protocol Reference](#)

Protocol which defines properties and methods which are required for all ViewControllers which will accommodate the role for onboarding in SOS.

[SOSOptions Class Reference](#)

The SOSOptions class allows you to configure the behavior of any SOSSessionManager session.

[SOSReconnectStatus Constants Reference](#)

Defines the discrete network reconnect status which can occur within an SOS Session.

[SOSScreenAnnotations Class Reference](#)

Class to handle screen annotations. This class implements the SOSWebRTCDelegate.

[SOSScreenSharing Class Reference](#)

Interface for configuring screen sharing behavior.

[SOSScreenSharingBaseViewController Class Reference](#)

The SOSScreenSharingBaseViewController serves as the base controller which manages interactions between the UI and SOS backend for the screen sharing phase of SOS.

[SOSSessionBaseViewController Class Reference](#)

The SOSScreenSharingBaseViewController serves as the base controller which manages interactions between the UI and SOS backend for the screen sharing phase of SOS.

[SOSSessionManager Class Reference](#)

The SOSSessionManager class is the main interface to the SOS framework.

[SOSSessionState Constants Reference](#)

Full list of Session states the SOS framework can exhibit.

[SOSSessionViewController Protocol Reference](#)

Protocol which defines properties and methods which are required for all ViewControllers which will accommodate the role for screen sharing in SOS.

[SOSStopReason Constants Reference](#)

Reasons provided to the `[SOSDelegate sos:didStopWithReason:error:]` event.

[SOSFramework String Constants](#)

List of string constants associated with the Service SDK SOS framework.

[SOSTelemetryLogStream Constants Reference](#)

List of Timer-specific logging types which can be tracked.

[SOSUIAgentStreamReceivable Protocol Reference](#)

The SOSUIAgentStreamReceivable protocol allows your view controller implementation to handle an agent video stream.

[SOSUILineDrawingReceivable Protocol Reference](#)

The SOSUILineDrawingReceivable protocol allows your view controller implementation to handle line drawings.

[SOSUIPhase Constants Reference](#)

Sets the Phase of the UI that will be replaced by the user

[SOSFramework String Constants](#)

List of string constants associated with the Service SDK SOS framework.

SOSAgentAvailability Class Reference

The SOSAgentAvailability class allows you to configure periodic polling against a single SOS deployment for your organization.

When the availability changes, an [\[SOSAgentAvailabilityDelegate agentAvailability:didChange:\]](#) event is fired.

– startPollingWithOrganizationId:deploymentId:liveAgentPod:

Initializes the agent polling. With the given credentials this will begin polling to determine agent availability. This can be leveraged to provide context to modify application UI depending on agent availability.

Declaration

```
- (void)startPollingWithOrganizationId:(NSString *)organizationId deploymentId:(NSString *)deploymentID liveAgentPod:(NSString *)liveAgentPod
```

Parameters

Name	Description
organizationId	The Salesforce organization id.
deploymentID	The unique id of the deployment for this session.
liveAgentPod	The hostname for the LiveAgent pod that your organization has been assigned.

Discussion

Currently we can only support polling a single deployment at a time.

Declared In

SOSAgentAvailability.h

– stopPolling

Discontinues polling operations. It is recommended that you stop polling in situations or views where no SOS functionality is appropriate/implemented.

Declaration

```
- (void)stopPolling
```

Declared In

SOSAgentAvailability.h

availabilityStatus

The current availability status.

Declaration

```
@property (nonatomic) SOSAgentAvailabilityStatusType availabilityStatus
```

Declared In

SOSAgentAvailability.h

CATEGORY: Delegate Management**– addDelegate:**

Adds an instance of an NSObject implementing the [SOSAgentAvailabilityDelegate](#) protocol to the list of delegates to notify.

Declaration

```
- (void)addDelegate:(id<SOSAgentAvailabilityDelegate>) delegate
```

Parameters

Name	Description
delegate	NSObject instance to add.

Declared In

SOSAgentAvailability.h

– removeDelegate:

Removes an instance of an NSObject implementing the [SOSAgentAvailabilityDelegate](#) protocol to the list of delegates to notify.

Declaration

```
- (void)removeDelegate:(id<SOSAgentAvailabilityDelegate>) delegate
```

Parameters

Name	Description
delegate	NSObject instance to remove.

Declared In

SOSAgentAvailability.h

SOSAgentAvailabilityDelegate Protocol Reference

Delegate protocol for SOSAgentAvailability

Implement this protocol in your classes to listen for availability change events from the [SOSAgentAvailability](#) class.

– agentAvailability:didChange:

Delegate method invoked when the [SOSAgentAvailability](#) status has changed.

Declaration

```
- (void)agentAvailability:(__weak id)agentAvailability  
didChange:(SOSAgentAvailabilityStatusType)availabilityStatus
```

Parameters

Name	Description
agentAvailability	The SOSAgentAvailability instance which fired the event.
availabilityStatus	The current SOSAgentAvailabilityStatusType .

Declared In

SOSAgentAvailability.h

– agentAvailability:didError:

Delegate method invoked when the [SOSAgentAvailability](#) polling has returned an error.

Declaration

```
- (void)agentAvailability:(__weak id)agentAvailability didError:(NSError *)error
```

Parameters

Name	Description
agentAvailability	SOSAgentAvailability instance which invoked the delegate method.
error	NSError instance describing the error.

Declared In

SOSAgentAvailability.h

SOSAgentAvailabilityStatusType Constants Reference

Enum for the Availability Status updates generated by the didChange: event

Definition

```
typedef NS_ENUM(NSInteger, SOSAgentAvailabilityStatusType ) {  
  
    SOSAgentAvailabilityStatusUnknown = 0,  
  
    SOSAgentAvailabilityStatusAvailable = 1,  
  
    SOSAgentAvailabilityStatusUnavailable = 2,  
  
};
```

Constants**SOSAgentAvailabilityStatusUnknown**

The base status of the agentAvailability. This is the first response you will see after you begin polling. This state is reset whenever polling is stopped.

SOSAgentAvailabilityStatusAvailable

An agent may be currently available to receive incoming SOS sessions. It is possible that an agent may become unavailable between polling, so this is only an indicator that it is possible that an agent can answer a call.

SOSAgentAvailabilityStatusUnavailable

No agents are currently available to receive SOS calls.

Declared In

SOSAgentAvailability.h

SOSCameraType Constants Reference

Sets the starting view for the current SOS Session.

Definition

```
typedef NS_ENUM(NSInteger, SOSCameraType ) {  
  
    SOSCameraTypeScreenSharing,  
  
    SOSCameraTypeFrontFacing,  
  
    SOSCameraTypeBackFacing,  
  
};
```

```
};
```

Constants

SOSCameraTypeScreenSharing

Screen sharing mode. This mode is for sharing the screen view with an agent. If you wish to use this as the starting camera you must ensure that `[SOSOptions featureClientScreenSharingEnabled]` has been set to `YES`. If it is not, then the session will return a `SOSInvalidOptionsCameraSettings` error.

SOSCameraTypeFrontFacing

Camera on the front of the device (selfie camera). This mode is for sharing the front-facing camera with an agent. If you wish to use this as the starting camera you must ensure that `[SOSOptions featureClientFrontCameraEnabled]` has been set to `YES`. If it is not, then the session will return a `SOSInvalidOptionsCameraSettings` error.

SOSCameraTypeBackFacing

Camera on the back of the device. This mode is for sharing the back-facing camera with an agent. If you wish to use this as the starting camera you must ensure that `[SOSOptions featureClientBackCameraEnabled]` has been set to `YES`. If it is not, then the session will return a `SOSInvalidOptionsCameraSettings` error.

See Also

- [SOSOptions](#)

Declared In

`SOSOptions.h`

SOSConnectingBaseViewController Class Reference

The `SOSConnectingBaseViewController` serves as the base controller which manages interactions between the UI and SOS backend for the connecting phase of SOS.

If you wish to replace the onboarding UI, your class must implement this base class.

All method overrides require a call back to super for SOS to function properly.

– `handleEndSession:`

Action performed when confirming an end session.

Declaration

```
- (IBAction)handleEndSession:(id) sender
```

Parameters

Name	Description
sender	the object which triggered the action.

Discussion

Warning: Executing this action will immediately trigger the end session cleanup. The responsibility of handling user confirmation remains as an implementation detail for the view controller.

Declared In

`SOSConnectingBaseViewController.h`

SOSConnectingViewController Protocol Reference

Protocol which defines properties and methods which are required for all ViewControllers which will accommodate the role for onboarding in SOS.

CATEGORY: Notifications

– initializingNotification

Message from the backend which indicates that the session is initializing.

Declaration

```
- (void)initializingNotification
```

Discussion

This method allows your implementation to present a custom message for this event. This is entirely optional as no input is required from the user.

Declared In

`SOSConnectingBaseViewController.h`

– waitingForAgentNotification

Message from the backend which indicates that the session has been placed in a queue and we are waiting for an agent to connect.

Declaration

```
- (void)waitingForAgentNotification
```

Discussion

This method allows your implementation to present a custom message for this event. This is entirely optional as no input is required from the user.

Declared In

`SOSConnectingBaseViewController.h`

– agentJoinedNotification:

Message from the backend which indicates that an agent is connecting to the session. Once the session has been fully established the session will move on to the next phase.

Declaration

```
- (void)agentJoinedNotification:(NSString *)name
```

Parameters

Name	Description
name	The name of the Agent

Discussion

This method allows your implementation to present a custom message for this event. This is entirely optional as no input is required from the user.

Declared In

`SOSConnectingBaseViewController.h`

SOSDelegate Protocol Reference

The SOSDelegate protocol provides information about the SOS session.

See [SOSSessionManager](#) for more information about SOS session management.

– sosDidStart:

Tells the delegate that an SOS session has started.

Declaration

```
- (void)sosDidStart:(SOSSessionManager *)sos
```

Parameters

Name	Description
sos	SOSSessionManager instance that invoked the delegate method.

Discussion

This method is executed once the SOS Session has begun to initialize a connection.

Declared In

`SOSSessionManager.h`

– sos:didStopWithReason:error:

Tells the delegate that an SOS session is stopping.

Declaration

```
- (void)sos:(SOSSessionManager *)sos didStopWithReason:(SOSStopReason)reason
error:(NSError *)error
```

Parameters

Name	Description
sos	SOSSessionManager instance that invoked the delegate method.
reason	SOSStopReason enum for why the session ended.
error	NSError instance returned if the session ended as the result of an error. Compare the error code to SOSErrorCode for details about the error. NOTE: Error is <code>nil</code> if the session ended cleanly.

Discussion

This event is invoked when the session is entering its cleanup phase.

Declared In

`SOSSessionManager.h`

– sosWillReconnect:

Tells the delegate that an attempt is being made to reconnect to an SOS session.

Declaration

```
- (void) sosWillReconnect: (SOSSessionManager *) sos
```

Parameters

Name	Description
sos	SOSSessionManager instance that invoked the delegate method.

Discussion

This is executed if the SOS session needs to reconnect.

Declared In

`SOSSessionManager.h`

– sosDidConnect:

Calls the delegate when the SOS session has connected. The session is now fully active.

Declaration

```
- (void) sosDidConnect: (SOSSessionManager *) sos
```


Parameters

Name	Description
sos	SOSSessionManager instance that invoked the delegate method.

Declared In

`SOSSessionManager.h`

– sos:didCreateSession:

Calls the delegate when the SOS session has been created.

Declaration

```
- (void)sos:(SOSSessionManager *)sos didCreateSession:(NSString *)sessionId
```

Parameters

Name	Description
sos	SOSSessionManager instance that invoked the delegate method.
sessionId	<code>NSString</code> of the sessionId for the session that has just started.

Declared In

`SOSSessionManager.h`

– sos:didError:

Tells the delegate that an error occurred during an active SOS session.

Declaration

```
- (void)sos:(SOSSessionManager *)sos didError:(NSError *)error
```

Parameters

Name	Description
sos	SOSSessionManager instance that invoked the delegate method.
error	<code>NSError</code> instance describing the error. Compare the error code to SOSErrorCode for details about the error.

Declared In

`SOSSessionManager.h`

– sos:stateDidChange:previous:

Tells the delegate that the SOS state changed.

Declaration

```
- (void)sos:(SOSSessionManager *)sos stateDidChange:(SOSSessionState)current
previous:(SOSSessionState)previous
```

Parameters

Name	Description
sos	SOSSessionManager instance that executed the delegate.
current	The new SOSSessionState that has been set on the SOSSessionManager instance.
previous	The previous SOSSessionState .

Declared In

[SOSSessionManager.h](#)

SOSErrorCode Constants Reference

SOS ErrorCode definitions.

Errors fall into one of several groups (or series) of errors.

If there is an error thrown by a library dependency it will be included as metadata in the userInfo of the error which is returned to the application code.

Definition

```
typedef NS_ENUM(NSInteger, SOSErrorCode ) {

    SOSGenericError = 1000,

    SOSInvalidOptions = 1001,

    SOSInvalidOptionsCameraSettings = 1002,

    SOSConnectionError = 2000,

    SOSSessionIsActiveError = 2001,

    SOSNoActiveSessionError = 2002,

    SOSNetworkUnavailableError = 2003,

    SOSServerError = 3000,

    SOSServerMetaDataError = 3001,
```

```
SOSServerSessionCreationError = 3002,  
  
SOSAgentAvailabilityError = 3003,  
  
SOSCommunicationError = 4000,  
  
SOSNoAgentsAvailableError = 4001,  
  
SOSNetworkTestError = 4002,  
  
SOSInsufficientNetworkError = 4003,  
  
SOSInternalError = 5000,  
  
SOSRTCProviderError = 6000,  
  
SOSRTCProviderAuthenticationError = 6001,  
  
};
```

Constants

SOSGenericError

Unclassified error. This results from an unknown or unexpected error state.

SOSInvalidOptions

Returned from [\[SOSSessionManager startSessionWithOptions:completion:\]](#) if the [SOSOptions](#) provided is `nil` or invalid.

SOSInvalidOptionsCameraSettings

Returned from [\[SOSSessionManager startSessionWithOptions:completion:\]](#) if [SOSOptions](#) contains invalid camera options. (e.g. `initialCameraType = SOSCameraTypeBackFacing` && [\[SOSOptions featureClientBackCameraEnabled\] == NO](#))

SOSConnectionError

Standard connection error. This can be thrown from any part of the SOS Session connection flow.

SOSSessionIsActiveError

Returned from [\[SOSSessionManager startSessionWithOptions:completion:\]](#) if there is a session already active or in progress.

SOSNoActiveSessionError

Returned from [\[SOSSessionManager stopSessionWithCompletion:\]](#) if there is no active session to stop.

SOSNetworkUnavailableError

Returned if the network becomes unavailable.

SOSServerError

Standard server error. This can be returned from operations made to the SOS server.

SOSServerMetadataError

Server metadata error. This can be returned from operations attempting to update the state of the session.

SOSServerSessionCreationError

Error returned when the session creation request is not successful.

SOSAgentAvailabilityError

Error returned when agent availability API encounters a problem.

SOSCommunicationError

Standard communication error. This can be returned from operations dependent on communication between SOS or any other remote system.

SOSNoAgentsAvailableError

Returned by the framework if there are no agents available to serve a session attempt.

SOSNetworkTestError

Returned by the framework if the network test has failed to start.

SOSInsufficientNetworkError

Returned by the framework if the network test has determined that the environment cannot support an SOS session.

SOSInternalError

Standard internal framework error. This can be returned from operations performed within the SOS framework.

SOSRTCProviderError

Standard RTC Provider error. This can be returned as a result of operations performed by the RTC provider.

SOSRTCProviderAuthenticationError

Error returned if there is a problem creating an authentication token with the WebRTC provider.

Declared In

`SOSError.h`

SOSLogger Class Reference

SOSLogging singleton which manages all logging messages generated by SOS.

+ sharedInstance

Singleton used to register your target to receive specific streams.

Declaration

```
+ (instancetype)sharedInstance
```

Declared In

`SOSLogger.h`

+ registerStream:target:

Register your target to receive streams. Only streams your target registers will be sent.

Declaration

```
+ (void)registerStream:(SOSLogStream)stream target:(__weak id)target
```

Parameters

Name	Description
stream	The stream the message will be logged to.
target	The object that stream will be registered to.

Declared In

SOSLogger.h

+ deregisterStream:target:

Unregister your target from the specified stream.

Declaration

```
+ (void)deregisterStream:(SOSLogStream) stream target:(__weak id) target
```

Parameters

Name	Description
stream	The message stream.
target	The object that stream will be unregistered from.

Declared In

SOSLogger.h

SOSLoggerProtocol Protocol Reference

Allows classes to listen to log messages generated by SOS.

– didReceiveStream:object:

Delegate method for when we receive stream events. Will be sent on a dispatch thread.

Declaration

```
- (void)didReceiveStream:(SOSLogStream) stream object:(id) object
```

Parameters

Name	Description
stream	The stream the message will be logged to.
object	The object to log. This may be a <code>NSString</code> or a <code>NSDictionary</code> , depending on the stream.

Declared In`SOSLogger.h`

SOSLogStream Constants Reference

List of Logging types which can be watched.

Definition

```
typedef NS_ENUM(NSInteger, SOSLogStream ) {

    SOSLogConnectivity = 0,

    SOSLogBackgrounded,

    SOSLogBattery,

    SOSLogData,

    SOSLogDevice,

    SOSLogFieldMasking,

    SOSLogLifeCycle,

    SOSLogNetworkTest,

    SOSLogConnectionStatus,

    SOSLogOrientation,

    SOSLogPhone,

    SOSLogSession,

    SOSLogError,

    SOSLogDebug,

    SOSLogAgentAvailability,

    SOSLogTelemetry,

    SOSLogAVSessionRef,

    SOSLogWarning,

    SOSLogRecording,

    SOSLogEvent,

};
```

Constants

SOSLogConnectivity

Log when the connectivity of the app changes.

SOSLogBackgrounded

Log when the app enters the background.

SOSLogBattery

Logging stream for battery.

SOSLogData

Logging stream for data.

SOSLogDevice

Log the device type and OS information.

SOSLogFieldMasking

Log when the field masking has turned on and off.

SOSLogLifeCycle

Logging the SOSLifeCycle changes.

SOSLogNetworkTest

Logging the network speed test.

SOSLogConnectionStatus

Logging the connection status during a session.

SOSLogOrientation

Logging stream for orientation.

SOSLogPhone

Logging the phone call info.

SOSLogSession

Logging stream for info.

SOSLogError

Logging stream for errors.

SOSLogDebug

Logging stream for debug, this is for local use only.

SOSLogAgentAvailability

Logging stream for agent availability.

SOSLogTelemetry

Logging stream for telemetry-based logs.

SOSLogAVSessionRef

Logging stream for AV Session reference id.

SOSLogWarning

Logging stream for warnings.

SOSLogRecording

Logging stream for session recording.

SOSLogEvent

Logging stream for events that occur.

Declared In

SOSLogger.h

SOSLogTimerCode Constants Reference

SOSLog Timer Code enum. See the schema for information:

<https://github.com/goinstant/schema/blob/master/schemas/common/messages/timer.json>

Definition

```
typedef NS_ENUM(NSInteger, SOSLogTimerCode ) {  
  
    SOSLogUserRequestToInqueue = 0,  
  
    SOSLogNetworkTestStartToNetworkTestEnd,  
  
    SOSLogInQueueToAgentAccept,  
  
    SOSLogAgentAcceptToConnected,  
  
    SOSLogCaptureScreenToCamera,  
  
    SOSLogCaptureCameraToScreen,  
  
    SOSLogCaptureSwapCamera,  
  
    SOSLogOnloadToWidgetReady,  
  
};
```

Constants**SOSLogUserRequestToInqueue**

userequest-to-inqueue

SOSLogNetworkTestStartToNetworkTestEnd

networkteststart-to-networktestend

SOSLogInQueueToAgentAccept

inqueue-to-agentaccept

SOSLogAgentAcceptToConnected

agentaccept-to-connected

SOSLogCaptureScreenToCamera

capture-screen-to-camera

SOSLogCaptureCameraToScreen

capture-camera-to-screen

SOSLogCaptureSwapCamera

capture-swap-camera

SOSLogOnloadToWidgetReady

onload-to-widget-ready

Declared In

SOSLogger.h

SOSMaskable Protocol Reference

Protocol for objects that can be registered as maskable. Anything implementing this protocol can be registered with the SOS Masking class, which will send the messages declared below at the appropriate times to enable/disable the masks during an SOS session.

– setMaskState:

Update the UI or do other processing to respond to a change in the masking requirements during an SOS session.

Declaration

```
- (void) setMaskState: (SOSMaskState) state
```

Parameters

Name	Description
state	The new state of the mask.

Declared In

SOSMaskable.h

SOSMaskedTextField Class Reference

Custom UITextField that auto-masks itself when screen sharing is enabled during an SOS session. Otherwise identical to the default UITextField.

– initWithFrame:

Initializes and returns a newly allocated SOSMaskedTextField object with the specified frame rectangle.

Declaration

```
- (id)initWithFrame:(CGRect) aRect
```

Parameters

Name	Description
aRect	The frame rectangle for the view, measured in points. The origin of the frame is relative to the superview in which you plan to add it. This method uses the frame rectangle to set the center and bounds properties accordingly.

Return Value

An initialized SOSMaskedTextField object or nil if the object couldn't be created.

Discussion

Create a SOSMaskedTextField in the given frame. This UIElement will be masked when the screen is shared with the agent view. When the user touches a SOSMaskedTextField while their screen is being shared, the mask will be removed and the screen will not be shared while the masked is removed.

If you used Interface Builder to design your interface, this method is not called when your view objects are loaded from the nib file. To add a SOSMaskedTextField in your nib just change the class of the UITextField to SOSMaskedTextField.

Declared In

SOSMaskedTextField.h

– initWithFrame:maskPattern:borderColor:text:textColor:

Initializes and returns a newly allocated SOSMaskedTextField object with the specified frame rectangle, maskPattern and borderColor.

Declaration

```
- (id)initWithFrame:(CGRect) aRect maskPattern:(NSString *)maskPattern borderColor:(UIColor *)borderColor text:(NSString *)text textColor:(UIColor *)textColor
```

Parameters

Name	Description
aRect	The frame rectangle for the view, measured in points. The origin of the frame is relative to the superview in which you plan to add it. This method uses the frame rectangle to set the center and bounds properties accordingly.
maskPattern	The file name for the mask color that will be created for the view. The image will to be used to create a UIColor using colorWithPatternImage:. If the value is nil the default mask pattern will be used.
borderColor	A UIColor that will create a border color around your SOSMaskedTextView. This is a highlight to the maskPattern image that will hid the contents of the SOSMaskedTextView. In the

Name	Description
	middle of the view will be a the word "Hidden" the back ground of this image will also be changed with the border color. If the value is nil the default mask pattern will be used.
text	The text that will be displayed in the middle of the masked text field. If this text needs to be localized ensure a localize string is passed in to this variable. If this variable is set to nil the default localized value of "Hidden" will be passed in.
textColor	The color of the text in the middle of the masked text field. If this variable is set to nil the default color will be white.

Return Value

An initialized SOSMaskedTextField object or nil if the object couldn't be created.

Discussion

Create a SOSMaskedTextField in the given frame with a custom masked image string, borderColor, text, and textColor. The UIElement will be masked when the screen is shared with the agent view. When the user touches a SOSMaskedTextField while their screen is being shared, the mask will be removed and the screen will not be shared while the masked is removed.

If you used Interface Builder to design your interface, this method is not called when your view objects are loaded from the nib file. To add a SOSMaskedTextField in your nib just change the class of the UITextField to SOSMaskedTextField.

Setting the MaskPattern, BorderColor, Text, and TextColor will adjust the look of the mask, this can be done to adjust the mask to the look and feel of the app. If you are using Interface Builder add following RunTime Attributes in the nib Identity Inspector maskingPattern, borderColor, maskText, and maskTextColor.

Declared In

SOSMaskedTextField.h

SOSMasking Class Reference

The SOSMasking class is responsible for managing the field masking applied during an SOS session. Sensitive fields or sections of the application may be masked to prevent them from appearing to the customer service agent in the video feed.

You can retrieve a reference to this class from the [SOSSessionManager](#) instance, and then use the APIs provided below to add or remove parts of your application to be masked on an as-needed basis.

CATEGORY: KVO-compliant properties

state

The current state of the masks. This state is driven by the state of the SOS session and SOS screen sharing.

Declaration

```
@property (readonly, atomic) SOSMaskState state
```

Declared In

SOSMasking.h

CATEGORY: Adding Masks

– registerView:

Register an object conforming to the [SOSMaskable](#) protocol to receive messages regarding the current masking [state](#).

Declaration

```
- (void)registerView:(id<SOSMaskable>) view
```

Parameters

Name	Description
view	The maskable object.

Discussion

This object is *not* retained by the SOSMasking class.

Declared In

SOSMasking.h

CATEGORY: Removing Masks

– registerDelegates

Register the delegates required for the session.

Declaration

```
- (void)registerDelegates
```

Declared In

SOSMasking.h

– removeDelegates

Remove the delegates required for the session.

Declaration

```
- (void)removeDelegates
```

Declared In

SOSMasking.h

SOSMaskState Constants Reference

Enumeration of the various states a given mask can be in.

Definition

```
typedef NS_ENUM(NSUInteger, SOSMaskState ) {  
  
    SOSMaskStateDisabled,  
  
    SOSMaskStateActive,  
  
    SOSMaskStateInactive,  
  
};
```

Constants

SOSMaskStateDisabled

No SOS session is active, so the mask is completely disabled.

SOSMaskStateActive

An SOS session is active and the screen is being shared; the mask must obscure the masked element.

SOSMaskStateInactive

An SOS session is active, but the screen is not currently being shared.

Declared In

SOSMaskable.h

SOSNetworkStatus Constants Reference

Defines the discrete network conditions which can occur within an SOS Session.

Definition

```
typedef NS_ENUM(NSUInteger, SOSNetworkStatus ) {  
  
    SOSNetworkStatusOk,  
  
    SOSNetworkStatusPoor,  
  
};
```

Constants

SOSNetworkStatusOk

The network and bandwidth tests run during a session have determined that the network is capable of sustaining a healthy SOS session.

SOSNetworkStatusPoor

The network and bandwidth tests run during a session have indicated that the network may not be able to fully support an SOS session. Audio and video artifacts may appear during the session. In the worst case the session itself could timeout.

Declared In

`SOSNetworkStatus.h`

SOSOnboardingBaseViewController Class Reference

The `SOSOnboardingBaseViewController` serves as the base controller which manages interactions between the UI and SOS backend for the onboarding phase of SOS.

If you wish to replace the onboarding UI, your class must implement this base class.

All method overrides require a call back to super for SOS to function properly.

CATEGORY: UI Actions

– `handleStartSession:`

Action performed when confirming a start session.

Declaration

```
- (IBAction)handleStartSession:(id) sender
```

Parameters

Name	Description
sender	The object which triggered the action.

Discussion

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSOnboardingBaseViewController.h`

– `handleCancel:`

Action performed when cancelling a session.

Declaration

```
- (IBAction)handleCancel:(id) sender
```

Parameters

Name	Description
sender	The object which triggered the action.

Discussion

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSOnboardingBaseViewController.h`

SOSOnboardingViewController Protocol Reference

Protocol which defines properties and methods which are required for all ViewControllers which will accommodate the role for onboarding in SOS.

CATEGORY: Alerts

– `willHandleConnectionPrompt`

This method determines whether this view controller will handle a connection request prompt. [Default YES] If this method returns NO then the SOS backend will directly proceed to the connection phase.

Declaration

```
- (BOOL)willHandleConnectionPrompt
```

Return Value

Whether this view controller will handle a connection prompt.

Discussion

Return NO if you do not wish to display a pre-connection onboarding screen for SOS sessions. If this method returns NO your class will not receive a [\[SOSOnboardingViewController connectionPromptRequested\]](#) call.

Declared In

`SOSOnboardingBaseViewController.h`

– `connectionPromptRequested`

This method will be executed on the SOSOnboardingViewController when the SOS backend has requested a user response. At this point it is appropriate to display a UI element which will allow a user to confirm or cancel.

Declaration

```
- (void)connectionPromptRequested
```

Discussion

The SOS backend will wait until it receives an appropriate response. Your UI element must trigger either: [\[SOSOnboardingBaseViewController handleStartSession:\]](#) or [\[SOSOnboardingBaseViewController handleCancel:\]](#)

See Also

- [\[SOSOnboardingViewController willHandleConnectionPrompt\]](#)

Declared In

`SOSOnboardingBaseViewController.h`

SOSOptions Class Reference

The SOSOptions class allows you to configure the behavior of any SOSSessionManager session.

This object determines how an SOS session is routed, as well as the behavior of a session once established.

CATEGORY: Initialization

+ optionsWithLiveAgentPod:orgId:deploymentId:

Instantiates an SOSOptions object for use with `[SOSSessionManager startSessionWithOptions:completion:]`.

Declaration

```
+ (instancetype)optionsWithLiveAgentPod:(NSString *)liveAgentPod orgId:(NSString *)orgId
deploymentId:(NSString *)deploymentId
```

Parameters

Name	Description
liveAgentPod	The hostname for the LiveAgent pod that your organization has been assigned. Also known as the Live Agent Endpoint.
orgId	The Salesforce organization ID.
deploymentId	The unique ID of the deployment for this session.

Return Value

An SOSOptions instance.

See Also

- `@property liveAgentPod`
- `@property orgId`
- `@property deploymentId`

Declared In

SOSOptions.h

– validate:

Validates the current values of the SOSOptions properties. The error parameter is nil on success. Otherwise, it contains a list of invalid properties.

Declaration

```
- (BOOL)validate:(NSError *__autoreleasing *)error
```

Parameters

Name	Description
error	An error object if there are invalid parameters.

Return Value

Whether the `SOSOptions` values are valid.

Declared In

`SOSOptions.h`

– setViewControllerClass:for:

Sets the replacement viewControllers for a phase of the UI interaction.

Declaration

```
- (void)setViewControllerClass:(Class)class for:(SOSUIPhase)phase
```

Parameters

Name	Description
class	A ViewController class that will replace the default ViewController UI.
phase	The Phase of the SOS session UI that is being replaced.

Declared In

`SOSOptions.h`

CATEGORY: SFDC Service Cloud / Live Agent Configuration**liveAgentPod**

The hostname for the LiveAgent pod that your organization has been assigned.

Declaration

```
@property (nonatomic) NSString *liveAgentPod
```

Discussion

To get this value, from Setup, search for `Live Agent Settings` and copy the hostname from the `Live Agent API Endpoint`.

Declared In

`SOSOptions.h`

orgId

Salesforce organization ID.

Declaration

```
@property (nonatomic) NSString *orgId
```

Discussion

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

Declared In

`SOSOptions.h`

deploymentId

The unique ID of the deployment for this session.

Declaration

```
@property (nonatomic) NSString *deploymentId
```

Discussion

A new deployment ID can be created in the Setup tab under `SOS Deployments`. The deployment ID is used to route specific types of support requests to any number of agents. You can set up any number of deployment IDs for your application to accommodate any routing model you choose. To get this value, from Setup, search for `SOS Deployments`, click the correct deployment and copy the `Deployment ID`.

Declared In

`SOSOptions.h`

CATEGORY: Session Management**sessionRetryTime**

The length of time (in seconds) before SOS prompts the user to retry or cancel.

Declaration

```
@property (nonatomic) NSTimeInterval sessionRetryTime
```

Declared In

`SOSOptions.h`

applicationVersion

By default, we send along the bundle version with session requests. If you wish to override this explicitly, you can use this property.

Declaration

```
@property (nonatomic) NSString *applicationVersion
```

Discussion

Warning: Note that only **64 MAX character strings** are supported.

Declared In

`SOSOptions.h`

initialCameraType

The initial `SOSCameraType` setting to be streamed upon starting a session.

Declaration

```
@property (nonatomic) SOSCameraType initialCameraType
```

Discussion

Default: `SOSCameraTypeScreenSharing`

Declared In

`SOSOptions.h`

remoteLoggingEnabled

Determines whether session logs are sent for collection. Logs sent remotely do not collect personal information. Unique IDs are created for tying logs to sessions, and those IDs cannot be correlated back to specific users.

Declaration

```
@property (nonatomic) BOOL remoteLoggingEnabled
```

Discussion

Default: `YES`

Declared In

`SOSOptions.h`

initialAgentVideoStreamActive

Whether the agent video stream is active upon starting a session.

Declaration

```
@property (nonatomic) BOOL initialAgentVideoStreamActive
```

Discussion

Default: `YES`

Declared In

`SOSOptions.h`

featureAgentVideoStreamEnabled

Whether the agent video stream is enabled for the session.

Declaration

```
@property (nonatomic) BOOL featureAgentVideoStreamEnabled
```

Discussion

Default: `YES`

Declared In

`SOSOptions.h`

featureClientScreenSharingEnabled

Whether screen sharing is enabled for the session.

Declaration

```
@property (nonatomic) BOOL featureClientScreenSharingEnabled
```

Discussion

Default: YES

Declared In

SOSOptions.h

featureClientFrontCameraEnabled

Whether the front-facing (selfie) camera is enabled for the session.

Declaration

```
@property (nonatomic) BOOL featureClientFrontCameraEnabled
```

Discussion

Default: NO

Declared In

SOSOptions.h

featureClientBackCameraEnabled

Whether the back-facing camera is enabled for the session.

Declaration

```
@property (nonatomic) BOOL featureClientBackCameraEnabled
```

Discussion

Default: NO

Declared In

SOSOptions.h

featureNetworkTestEnabled

Whether the network test is enabled before and during a session.

Declaration

```
@property (nonatomic) BOOL featureNetworkTestEnabled
```

Discussion

Default: YES

Declared In

SOSOptions.h

customFieldData

NSMutableDictionary that can be used to send custom data.

Declaration

```
@property (nonatomic) NSMutableDictionary *customFieldData
```

Declared In

SOSOptions.h

viewControllerClasses

NSDictionary that contains the registered viewControllers that will override the internal user interfaces

Declaration

```
@property (retain, nonatomic) NSDictionary *viewControllerClasses
```

Declared In

SOSOptions.h

initialAgentStreamPosition

CGPoint that contains the starting center location of the session controls view. If using replacement UI, this functionality needs to be implemented in the replacement UI code.

Declaration

```
@property (assign, nonatomic) CGPoint initialAgentStreamPosition
```

Declared In

SOSOptions.h

SOSReconnectStatus Constants Reference

Defines the discrete network reconnect status which can occur within an SOS Session.

Definition

```
typedef NS_ENUM(NSUInteger, SOSReconnectStatus ) {  
  
    SOSNetworkReconnectStarted,  
  
    SOSNetworkReconnectFinished,  
  
};
```

Constants

SOSNetworkReconnectStarted

The a network reconnect implemented by the WebRTC provider has signaled a reconnect has started.

SOSNetworkReconnectFinished

The network reconnect implemented by the WebRTC provider has signaled a reconnect has finished.

Declared In

SOSNetworkStatus.h

SOSScreenAnnotations Class Reference

Class to handle screen annotations. This class implements the SOSWebRTCDelegate.

enabled

If enabled, will allow drawing on the current screen by the service agent.

Declaration

```
@property (atomic) BOOL enabled
```

Discussion

Default: YES

NOTE: if you set this to NO while screen sharing is active, it will execute [\[SOSScreenAnnotations stop\]](#).

Declared In

SOSScreenAnnotations.h

clearOnTouches

When [enabled](#) all touch events will clear the screen of annotations, unless the Agent is currently drawing. If you wish to handle clearing annotations yourself set this to NO.

Declaration

```
@property (atomic) BOOL clearOnTouches
```

Discussion

Default: YES

Declared In

SOSScreenAnnotations.h

lineColor

Customizable color of the line drawing feature. Will not change the color of existing lines on the screen.

Declaration

```
@property (nonatomic) UIColor *lineColor
```

Discussion

Default: [UIColor redColor]

Declared In`SOSScreenAnnotations.h`**lineWidth**

Customizable width of the line drawing feature. Does not change the width of existing lines on the screen.

Declaration

```
@property (nonatomic) CGFloat lineWidth
```

Discussion

Default: 5.0f

Declared In`SOSScreenAnnotations.h`**– clearDraw**

Clears the current annotation by reinitializing the drawer.

Declaration

```
- (void)clearDraw
```

Declared In`SOSScreenAnnotations.h`**– start**

Initiates the drawing feature.

Declaration

```
- (void)start
```

Declared In`SOSScreenAnnotations.h`**– stop**

Stops the drawing feature.

Declaration

```
- (void)stop
```

Declared In`SOSScreenAnnotations.h`

SOSScreenSharing Class Reference

Interface for configuring screen sharing behavior.

enabled

If enabled, will allow sharing of the current screen context to a connected service agent. Default: `YES`

Declaration

```
@property (atomic) BOOL enabled
```

Declared In

`SOSScreenSharing.h`

SOSScreenSharingBaseViewController Class Reference

The `SOSScreenSharingBaseViewController` serves as the base controller which manages interactions between the UI and SOS backend for the screen sharing phase of SOS.

If you wish to replace the screen sharing UI, your class must implement this base class.

All method overrides require a call back to super for SOS to function properly.

– `handleCameraTransition:`

This will trigger a transition to the camera view, and mark the SOS Session as being in the camera phase. At this point the screen sharing phase is ending, and the view controller will attempt to clean up.

Declaration

```
- (IBAction)handleCameraTransition:(id) sender
```

Parameters

Name	Description
sender	The object which triggered the action.

Discussion

Warning: Please ensure that your view controller cleans up entirely. Any reference cycles will result in additional instances of this view controller in memory when returning to the screen sharing phase.

This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSScreenSharingBaseViewController.h`

CATEGORY: Session State Management

– `setAgentStreamCenter:`

Required method to call if you wish to have the agent stream container on the service cloud widget move as a response to changes in your UI. The center point you provide will be used to update the agent container position in the service cloud widget.

Declaration

```
- (void)setAgentStreamCenter:(CGPoint) center
```

Parameters

Name	Description
center	The new center point of the agent container in the service cloud widget.

Discussion

Warning: Each time you call this method it will result in a status update on the session. For that reason it is recommended that you do not call this method during a drag gesture, but rather update the position once the position has been determined.

This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSScreenSharingBaseViewController.h`

– screenSharingStateDidChange:

Required method to call if you wish to have a UI indicator that informs the user of changes to the screen sharing state. This method will be called when the screenSharing state changes.

Declaration

```
- (void)screenSharingStateDidChange:(BOOL) sharing
```

Parameters

Name	Description
Sharing	The current sharing state of the session.

Declared In

`SOSScreenSharingBaseViewController.h`

SOSSessionBaseViewController Class Reference

The `SOSScreenSharingBaseViewController` serves as the base controller which manages interactions between the UI and SOS backend for the screen sharing phase of SOS.

If you wish to replace the screen sharing UI, your class must implement this base class.

All method overrides require a call back to `super` for SOS to function properly.

CATEGORY: UI Action Handlers

– `handlePause:`

Action performed when pausing or unpausing an SOS Session.

Declaration

```
- (IBAction)handlePause:(id) sender
```

Parameters

Name	Description
sender	The object which triggered the action.

Discussion

This method will automatically call [\[SOSSessionBaseViewController setSOSPaused:\]](#) The value will be derived from the negation of [\[SOSSessionBaseViewController isSOSPaused\]](#) If you wish to handle the muted state in a different way, you can ignore this action and use [\[SOSSessionBaseViewController setSOSPaused:\]](#) directly instead.

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSSessionBaseViewController.h`

– `handleMute:`

Action performed when muting or unmuting the audio for an SOS Session.

Declaration

```
- (IBAction)handleMute:(id) sender
```

Parameters

Name	Description
sender	The object which triggered the action.

Discussion

This method will automatically call [\[SOSSessionBaseViewController setSOSMuted:\]](#) The value will be derived from the negation of [\[SOSSessionBaseViewController isSOSMuted\]](#) If you wish to handle the muted state in a different way, you can ignore this action and use [\[SOSSessionBaseViewController setSOSMuted:\]](#) directly instead.

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In`SOSSessionBaseViewController.h`**– handleEndSession:**

Action performed when confirming an end session.

Declaration

```
- (IBAction)handleEndSession:(id) sender
```

Parameters

Name	Description
sender	The object which triggered the action.

Discussion

Warning: Executing this action will immediately trigger the end session cleanup. The responsibility of handling user confirmation remains as an implementation detail for the view controller.

This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In`SOSSessionBaseViewController.h`**CATEGORY: Session State Management****– setSOSMuted:**

Call this (required) method when informing the SOS backend that you wish to mute outgoing audio. This method will instruct the backend to continue/discontinue transmitting audio from the device.

Declaration

```
- (void)setSOSMuted:(BOOL) isMuted
```

Parameters

Name	Description
isMuted	Whether the outgoing audio stream has been muted.

Discussion

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In`SOSSessionBaseViewController.h`

– isSOSMuted

Call this (required) method to get the current muted state of the SOS Session.

Declaration

```
- (BOOL) isSOSMuted
```

Return Value

The current muted state of the SOS Session.

Discussion

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSSessionBaseViewController.h`

– setSOSPaused:

Call this (required) method when informing the SOS backend that you wish to pause the session. This method will instruct the backend to continue/discontinue all audio and video traffic from the device.

Declaration

```
- (void) setSOSPaused: (BOOL) isPaused
```

Parameters

Name	Description
<code>isPaused</code>	Whether the outgoing audio/video traffic has been paused.

Discussion

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSSessionBaseViewController.h`

– isSOSPaused

Call this (required) method to get the current local paused state of the SOS Session.

Declaration

```
- (BOOL) isSOSPaused
```

Return Value

The current paused state of the SOS Session.

Discussion

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSSessionBaseViewController.h`

– isRecording

This value is set on the service cloud integration. It should not change during the lifetime of a session. You can use this method to determine if you wish to display a recording status in your UI.

Declaration

```
- (BOOL)isRecording
```

Return Value

Whether the session is being recorded.

Discussion

Warning: This method is used to communicate between the UI API and the SOS backend. If you wish to override this method you can do so safely; however you must call this method on `super`. Failure to do so will result in undefined and likely broken behavior from SOS.

Declared In

`SOSSessionBaseViewController.h`

SOSSessionManager Class Reference

The `SOSSessionManager` class is the main interface to the SOS framework.

This object manages the flow of SOS sessions throughout the lifetime of the app. Configuration and customization of the SOS framework is handled through the public properties on the `SOSSessionManager` instance.

`SOSSessionManager` conforms to a multicast delegate model for messaging. Any class which implements the `SOSDelegate` protocol can be added to a list of delegates to receive messages asynchronously.

CATEGORY: SOS Management Objects

annotations

Public reference to the `SOSScreenAnnotations` instance used by the SOS framework.

Declaration

```
@property (readonly, atomic) SOSScreenAnnotations *annotations
```

Declared In

`SOSSessionManager.h`

masking

Public reference to the [SOSMasking](#) instance used by the SOS framework.

Declaration

```
@property (readonly, atomic) SOSMasking *masking
```

Declared In

SOSSessionManager.h

agentAvailability

Public reference to the [SOSAgentAvailability](#) object used for checking availability of SOS agents.

Declaration

```
@property (readonly, atomic) SOSAgentAvailability *agentAvailability
```

Declared In

SOSSessionManager.h

CATEGORY: Other Properties

state

The current [SOSSessionState](#).

Declaration

```
@property (readonly, atomic) SOSSessionState state
```

Declared In

SOSSessionManager.h

CATEGORY: Session Management

– startSessionWithOptions:

Starts the process for creating an SOS session.

Declaration

```
- (void)startSessionWithOptions:(SOSOptions *)options
```

Parameters

Name	Description
options	SOSOptions object that determines the behavior of this session.

Discussion

Equivalent to invoking `startSessionWithOptions:completion:` and providing a `nil` block.

See Also

- `- startSessionWithOptions:completion:`
- `SOSOptions`

Declared In

`SOSSessionManager.h`

– startSessionWithOptions:completion:

Starts the process for creating an SOS session.

Declaration

```
- (void)startSessionWithOptions:(SOSOptions *)options
completion:(SOSCompletionHandler)block
```

Parameters

Name	Description
options	<code>SOSOptions</code> object that determines the behavior of this session.
block	Completion block which is executed when the session has been fully connected to all services. NOTE: At this point the session is active and waiting for an agent to join. NOTE: The <code>NSError</code> returned in the block is <code>nil</code> on success.

Discussion

By default, the user is asked if they would like to initiate an SOS session.

Delegate Methods:

- `[SOSDelegate sosDidConnect:]` Invoked when all session handshakes and negotiations have completed. The session is fully connected now.
- `[SOSDelegate sosDidStart:]` Invoked when the session has begun negotiating with required services.
- `[SOSDelegate sos:stateDidChange:previous:]` Invoked at each `state` change.
- `[SOSDelegate sos:didError:]` Invoked if there is an error at any point during the lifetime of a session.

See Also

- `SOSDelegate`
- `SOSOptions`

Declared In

`SOSSessionManager.h`

– stopSession

Begins a teardown of an SOS Session.

Declaration

```
- (void)stopSession
```

Discussion

Equivalent to invoking `stopSessionWithCompletion:` and providing a `nil` block.

See Also

- [– stopSessionWithCompletion:](#)

Declared In

`SOSSessionManager.h`

– stopSessionWithCompletion:

Begins a teardown of an SOS Session.

Declaration

```
- (void)stopSessionWithCompletion: (SOSCompletionHandler)block
```

Parameters

Name	Description
block	Completion block which is executed when the session has fully stopped, and all connected services have been torn down.

Discussion

By default, the user is asked if they would like to stop the SOS session.

Delegate Methods:

- [\[SOSDelegate sos:didStopWithReason:error:\]](#) Invoked when the session has been fully torn down.
- [\[SOSDelegate sos:stateDidChange:previous:\]](#) Invoked at each [state](#) change.
- [\[SOSDelegate sos:didError:\]](#) Invoked if there is an error at any point during the lifetime of a session.

NOTE: The `NSError` returned in the block is `nil` on success.

See Also

- [SOSDelegate](#)

Declared In

`SOSSessionManager.h`

screenSharing

Provides a reference to an instance of the [SOSScreenSharing](#) class.

Declaration

```
@property (readonly, atomic) SOSScreenSharing *screenSharing
```

Declared In

SOSSessionManager.h

CATEGORY: Delegate Management**– addDelegate:**

Adds an instance of an `NSObject` implementing the [SOSDelegate](#) protocol to the list of delegates to notify.

Declaration

```
- (void)addDelegate:(id<SOSDelegate>) delegate
```

Parameters

Name	Description
delegate	<code>NSObject</code> instance to add.

Declared In

SOSSessionManager.h

– removeDelegate:

Removes an instance of an `NSObject` implementing the [SOSDelegate](#) protocol to the list of delegates to notify.

Declaration

```
- (void)removeDelegate:(id<SOSDelegate>) delegate
```

Parameters

Name	Description
delegate	<code>NSObject</code> instance to remove.

Declared In

SOSSessionManager.h

CATEGORY: Class Methods**+ frameworkVersion**

Returns the `NSString` containing the current version of the framework.

Declaration

```
+ (NSString *)frameworkVersion
```

Return Value

An NSString containing the framework version.

Declared In

SOSSessionManager.h

SOSSessionState Constants Reference

Full list of Session states the SOS framework can exhibit.

Definition

```
typedef NS_ENUM(NSInteger, SOSSessionState ) {  
  
    SOSSessionStateInactive = 0,  
  
    SOSSessionStateConfiguring = 1,  
  
    SOSSessionStateInitializing = 2,  
  
    SOSSessionStateConnecting = 3,  
  
    SOSSessionStateActive = 4,  
  
};
```

Constants**SOSSessionStateInactive**

No active session. There will be no outgoing/incoming SOS traffic.

SOSSessionStateConfiguring

Session is doing pre-initialization configuration steps, such as network testing.

SOSSessionStateInitializing

Session state is initializing and preparing to connect.

SOSSessionStateConnecting

Session is attempting a connection to a live agent.

SOSSessionStateActive

Live agent has connected and the session is fully active.

Declared In

SOSSessionManager.h

SOSSessionViewController Protocol Reference

Protocol which defines properties and methods which are required for all ViewControllers which will accommodate the role for screen sharing in SOS.

CATEGORY: Notifications

– reconnectingNotification

Message from the backend that the session is attempting to reconnect. You may receive this message in the event that the session is attempting to recover from a network disconnection or in the event of backgrounding the application or after receiving a phone call.

Declaration

```
- (void)reconnectingNotification
```

Discussion

This method allows your implementation to present a custom message for this event. This is entirely optional as no input is required from the user.

Declared In

`SOSSessionBaseViewController.h`

– reconnectionCompleteNotification

Message from the backend that the session has successfully reconnected.

Declaration

```
- (void)reconnectionCompleteNotification
```

Discussion

This method allows your implementation to present a custom message for this event. This is entirely optional as no input is required from the user.

Declared In

`SOSSessionBaseViewController.h`

– agentPausedConnectionNotification:

Message from the backend that the agent has paused/unpaused their connection. This method will be executed any time the agent pause state has changed.

Declaration

```
- (void)agentPausedConnectionNotification:(BOOL)paused
```

Parameters

Name	Description
paused	The current agent paused state. YES means that the agent is currently pausing their connection.

Declared In

`SOSSessionBaseViewController.h`

– networkStatusUpdateNotification:

Message from the backend that the network connection status has changed. The status is determined by bandwidth tests which are conducted during an SOS session. When the status changes you will receive this method with the current network status.

Declaration

```
- (void)networkStatusUpdateNotification:(SOSNetworkStatus) networkStatus
```

Parameters

Name	Description
<code>networkStatus</code>	The current network status.

Declared In

`SOSSessionBaseViewController.h`

– reconnectStatusUpdateNotification:

Message from the backend that the WebRTC provider has changed the reconnect status. The status message is determined by signals from the WebRTC provider. When the status changes, you will receive this method with the current reconnect status.

Declaration

```
- (void)reconnectStatusUpdateNotification:(SOSReconnectStatus) reconnectStatus
```

Parameters

Name	Description
<code>reconnectStatus</code>	The current network reconnect status.

Declared In

`SOSSessionBaseViewController.h`

SOSStopReason Constants Reference

Reasons provided to the `[SOSDelegate sos:didStopWithReason:error:]` event.

Definition

```
typedef NS_ENUM(NSInteger, SOSStopReason) {
    SOSStopReasonUserDisconnected = 1,
    SOSStopReasonAgentDisconnected = 2,
```

```

    SOSStopReasonSessionError = 3,

    SOSStopReasonExternalUnknown = 4,

    SOSStopReasonSessionTimeout = 5,

    SOSStopReasonBackgroundedBeforeConnected = 6,

    SOSStopReasonInvalid = -1,

};

```

Constants

SOSStopReasonUserDisconnected

User disconnected the session.

SOSStopReasonAgentDisconnected

Agent disconnected the session.

SOSStopReasonSessionError

Session ended due to an error.

SOSStopReasonExternalUnknown

Session was ended in response to the application attempting to terminate.

SOSStopReasonSessionTimeout

Session failed due to timeout.

SOSStopReasonBackgroundedBeforeConnected

Session ended because the app was backgrounded before the connection completed.

SOSStopReasonInvalid

Reset the cause for session disconnection.

Declared In

`SOSSessionManager.h`

SOSFramework String Constants

List of string constants associated with the Service SDK SOS framework.

Table 13: String Constants

String Token	Description	Default Value
ServiceCloud.SOS.Title	Dialogs *//* The title we present for any UI element that has a title, such as one of our dialogs	Salesforce SOS

String Token	Description	Default Value
ServiceCloud.SOS.ConnectPrompt	The dialog body we present when prompting to start an SOS session	You are about to start an SOS session. You will be able to talk and screen share with an Agent in real time. Are you ready to experience the future of customer service?
ServiceCloud.SOS.Session.Disconnect	The dialog body we present when confirming that a user would like to end their session (after hitting an end session button)	Press OK to disconnect from the session.
ServiceCloud.SOS.End.Session.Remote	The dialog body we present when the agent has ended the SOS session	The Agent has ended the SOS session.
ServiceCloud.SOS.End.Session.AgentMissing	The dialog body we present when the agent disconnects for an unknown reason	The agent may have disconnected. Press Quit to end the session now or Cancel to continue waiting.
ServiceCloud.SOS.Continue.Waiting.Prompt	The dialog body we present when asking a customer if they would like to continue waiting for a session	All agents are currently busy, do you wish to continue waiting?
ServiceCloud.SOS.End.Session.Timeout	The dialog body we present when we cannot start a session due to no agents being online"	Your SOS Session has timed out. Please try again.
ServiceCloud.SOS.End.Session.InsufficientNetwork	The dialog body we present when we cannot start a session due to insufficient network conditions	Network not able to support SOS, please try again later.
ServiceCloud.SOS.Network.Test.Error	The dialog body we present if the network test encounters an error	We are unable to conduct a network test. Unfortunately we cannot guarantee an awesome SOS customer experience. Would you like to continue anyway?
ServiceCloud.SOS.End.Session.No.Agent	The dialog body we present if no agents are current available	No agents are currently available, please try again later.
ServiceCloud.SOS.End.Session.Network.Lost	The dialog body we present if the network no longer functions	Your session has ended because the network has become unavailable. Please try again.
ServiceCloud.SOS.Disconnect.BackgroundedWhileConnecting	The dialog body we present if the app was backgrounded before the connection process finished	Your session has ended because the app was backgrounded before the session was fully established. Please try again.
ServiceCloud.SOS.Disconnect.Reconnect.Failure	The dialog body we present is the reconnect of a session fails	Failed to re-establish to the SOS session. Please try again.
ServiceCloud.SOS.Connect.Positive	The positive choice for the start session prompt (saying yes to the dialog)	OK
ServiceCloud.SOS.Connect.Connect	A positive choice for connecting to a session. This is shown in a session prompt	Connect

String Token	Description	Default Value
ServiceCloud.SOS.Connect.Negative	The negative choice for the start session prompt (saying no to the dialog)	Cancel
ServiceCloud.SOS.Disconnect.EndNow	A button message in a prompt to end a session now	End Now
ServiceCloud.SOS.Continue.Waiting.Negative	The negative choice for the continue waiting prompt (saying no to the dialog)	Quit
ServiceCloud.SOS.Session.Is.Agent.Joining	The notification message used when the agent is in the process of joining the SOS session	Agent now joining...
ServiceCloud.SOS.Session.Is.Starting	The notification message used when a session is starting up	Session is starting...
ServiceCloud.SOS.Session.Is.Initializing	The notification message used when a session is initializing	Initializing...
ServiceCloud.SOS.Session.Is.Waiting.For.Agent	The notification message used when the customer is ready to start a session, but still waiting for the agent to join	Waiting for an agent...
ServiceCloud.SOS.Session.Is.Reconnecting.Audio	The notification message used when the audio needs to be reconnected	Reconnecting Audio...
ServiceCloud.SOS.Network.Test.Is.Initializing	The notification message used when the network is initializing	Initializing Network...
ServiceCloud.SOS.Session.Is.Reconnecting.Network	The notification message used when a switching of networks has occurred and reconnect is required	Switching network, reconnecting...
ServiceCloud.SOS.UI.Status.Queued	Non-blocking UI status message displayed when the session is in the agent queue	In queue
ServiceCloud.SOS.UI.Status.Connecting	Status message displayed when the session is connecting to an agent	Connecting
ServiceCloud.SOS.UI.Status.Audio.Muted	Status message displayed when the user's audio is muted	Mic muted
ServiceCloud.SOS.UI.Status.Poor.Network	Status message displayed when network connection is poor during a session	Poor network
ServiceCloud.SOS.Session.Is.Paused	Status message displayed when agent video is paused	Session paused
ServiceCloud.SOS.Agent.Is.Paused	Status message displayed when the agent pauses the session	Session on hold
ServiceCloud.SOS.Session.Is.Paused.Alt	Status message displayed as a second message when the video is paused	(agent cannot see screen)

String Token	Description	Default Value
ServiceCloud.SOS.Agent.Is.Paused.Alt	Status message displayed as a second message when the agent pauses the session	(agent has paused session)
ServiceCloud.SOS.Disconnect.Prompt	Status message displayed when the disconnect yes or no options are shown	Disconnect from the session?
ServiceCloud.SOS.All.Agent.Busy	Status message displayed when all the agents are busy timeout is shown	All Agents are currently busy: Continue waiting?
ServiceCloud.SOS.Recording	Agent View */ /* Text overlayed on the agent container when the SOS session is being recorded	recording
ServiceCloud.SOS.Network.Test.Fail	Message that informs the user a network test is not possible	Unable to complete network test. Proceed anyways?
ServiceCloud.SOS.Network.Test.UnableToConnect	Message that informs the user the session could not connect	Unable to connect... Try again?
ServiceCloud.SOS.Session.Agent.Joined	The notification message used when an agent has joined and the app is backgrounded	Agent now joining, click here to return to the app
ServiceCloud.SOS.Onboarding.Header.Label	OnBoarding */ /* The header text shown at the top of the onboarding UI during session start.	SOS
ServiceCloud.SOS.Onboarding.Carousel.Intro.Text	The text displayed on the first default card of the onboarding dialog.	You are about to co-browse your app with an expert who can see and draw on your screen. Don't worry, they can't see you :)
ServiceCloud.SOS.Onboarding.Carousel.Dragging.Text	The text displayed on the second default card of the onboarding dialog.	Drag the SOS window anywhere on your screen or tap to reveal controls.
ServiceCloud.SOS.Onboarding.Carousel.Controls.Text	The text displayed on the third default card of the onboarding dialog.	Tap to reveal controls to mute mic, pause screen share, and end session.
ServiceCloud.SOS.Onboarding.Accept.Button.Label	The button text displayed on the onboarding UI's Accept button	OK, GOT IT
ServiceCloud.SOS.Video.Pause.Label	Session Controls Camera View */ /* The label under the pause/resume button in the video activity when the session is NOT paused.	Pause
ServiceCloud.SOS.Video.Resume.Label	The label under the pause/resume button in the video activity when the session is PAUSED.	Resume
ServiceCloud.SOS.Video.Unmute.Label	The label under the mute/unmute button in the video activity when the session is MUTED.	Unmute

String Token	Description	Default Value
ServiceCloud.SOS.Video.Mute.Label	The label under the mute/unmute button in the video activity when the session is NOT muted.	Mute
ServiceCloud.SOS.Video.End.Label	The label under the end session button in the video activity.	End
ServiceCloud.SOS.Video.Gesture.Info.Camera.Swap	Camera View *//* Text displayed on the Video UI to describe the camera swap gesture.	Double tap to flip camera
ServiceCloud.SOS.Video.Gesture.Info.Flashlight	Text displayed on the Video UI to describe the flashlight gesture.	Tap & hold anywhere to toggle flashlight
ServiceCloud.SOS.Video.Gesture.Info.Dismiss	Text displayed on the dismissal button for gesture information on the Video UI.	GOT IT
ServiceCloud.SOS.Video.Back.To.App.Label	Header that appears in the video activity that describes to the user they can click the header to return to the app.	Back to app
ServiceCloud.SOS.Video.Flashlight.On.Label	Label that describes a flashlight icon. Indicates that the phone flashlight is on	Flashlight On
ServiceCloud.SOS.Video.Muted.Label	Message that informs the user that the device has been muted	Muted
ServiceCloud.SOS.Video.Session.Pause.Header	The header of the message shown to the user when the session is paused in camera mode	Session paused
ServiceCloud.SOS.Video.Agent.Pause.Header	The header of the message shown to the user when the agent is paused in camera mode	Session on hold
ServiceCloud.SOS.Video.Session.Pause.Info	The text that appears below the header whenever the session is paused in camera mode	No one can see or hear you
ServiceCloud.SOS.Video.Agent.Pause.Info	The text that appears below the header whenever the agent is paused in camera mode	Agent has paused session
ServiceCloud.SOS.Video.Connecting.Label	Message that informs the user that the camera view is connecting	Connecting...

SOSTelemetryLogStream Constants Reference

List of Timer-specific logging types which can be tracked.

Definition

```
typedef NS_ENUM(NSInteger, SOS telemetryLogStream ) {  
  
    SOS telemetryLogStreamUserConfirmation = 0,  
  
    SOS telemetryLogStreamNetworkTest,  
  
    SOS telemetryLogStreamWaiting,  
  
    SOS telemetryLogStreamJoining,  
  
    SOS telemetryLogStreamConnected,  
  
};
```

Constants

SOSTelemetryLogStreamUserConfirmation

Telemetry stream for user confirmation.

SOSTelemetryLogStreamNetworkTest

Telemetry stream for network tests.

SOSTelemetryLogStreamWaiting

Telemetry stream for waiting in the LA queue.

SOSTelemetryLogStreamJoining

Telemetry stream for when joining a session.

SOSTelemetryLogStreamConnected

Telemetry stream for being connected to a session.

Declared In

`SOSLogger.h`

SOSUIAgentStreamReceivable Protocol Reference

The `SOSUIAgentStreamReceivable` protocol allows your view controller implementation to handle an agent video stream.

– `willHandleAgentStream`

Asks your delegate whether it can handle an agent video stream.

Declaration

```
- (BOOL)willHandleAgentStream
```

Return Value

`YES` if the class can handle agent streams.

Declared In`SOSUIAgentStreamReceivable.h`**– willHandleRemoteMovement**

Asks your delegate whether it can handle remote movement of the video stream.

Declaration

```
- (BOOL)willHandleRemoteMovement
```

Return Value

YES if the class can handle remote movement.

Declared In`SOSUIAgentStreamReceivable.h`**– willHandleAudioLevel**

Asks your delegate whether it can handle audio level adjustments.

Declaration

```
- (BOOL)willHandleAudioLevel
```

Return Value

YES if the class can handle audio levels.

Declared In`SOSUIAgentStreamReceivable.h`**– didReceiveAgentStreamView:**

Tells your delegate when there is an agent video stream.

Declaration

```
- (void)didReceiveAgentStreamView: (__weak UIView *)agentStreamView
```

Parameters

Name	Description
agentStreamView	The view containing the agent stream.

Declared In`SOSUIAgentStreamReceivable.h`**– didReceiveAgentStreamScreenCoordinate:**

Tells your delegate the new location coordinates of an agent stream.

Declaration

```
- (void) didReceiveAgentStreamScreenCoordinate: (CGPoint) coordinate
```

Parameters

Name	Description
<code>coordinate</code>	The new location coordinates.

Declared In

`SOSUIAgentStreamReceivable.h`

– didReceiveAudioLevelUpdate:

Tells your delegate when the audio level has changed.

Declaration

```
- (void) didReceiveAudioLevelUpdate: (CGFloat) audioLevel
```

Parameters

Name	Description
<code>audioLevel</code>	The new audio level.

Declared In

`SOSUIAgentStreamReceivable.h`

– didReceiveRecordingUpdate:

Tells your delegate when the recording of the session has changed.

Declaration

```
- (void) didReceiveRecordingUpdate: (BOOL) recording
```

Parameters

Name	Description
<code>recording</code>	The recording value.

Declared In

`SOSUIAgentStreamReceivable.h`

SOSUILineDrawingReceivable Protocol Reference

The `SOSUILineDrawingReceivable` protocol allows your view controller implementation to handle line drawings.

– willHandleLineDrawing

Asks the delegate whether it can handle line drawings.

Declaration

```
- (BOOL)willHandleLineDrawing
```

Return Value

YES if the class can handle line drawings.

Declared In

SOSUILineDrawingReceivable.h

– didReceiveLineDrawView:

Tells the delegate when a line is drawn on the screen.

Declaration

```
- (void)didReceiveLineDrawView:(__weak UIView *)drawView
```

Parameters

Name	Description
drawView	The view containing the drawing.

Declared In

SOSUILineDrawingReceivable.h

SOSUIPhase Constants Reference

Sets the Phase of the UI that will be replaced by the user

Definition

```
typedef NS_ENUM(NSInteger, SOSUIPhase ) {
    SOSUIPhaseOnboarding,
    SOSUIPhaseConnecting,
    SOSUIPhaseScreenSharing,
};
```

Constants

SOSUIPhaseOnboarding

Onboarding Phase. This phase is used for showing the user information about the support system that is about to be activated. The options could be presented to the user to allow access to the camera or mic. How the SOS system functions could also be relayed

to this user. The class associated with this phase must extend [SOSOnboardingBaseViewController](#) and implement the protocol [SOSOnboardingViewController](#).

SOSUIPhaseConnecting

Connecting Phase. This phase is used for showing the user information while the SOS system is connecting to the user. The class associated with this phase must extend [SOSConnectingBaseViewController](#) and implement the protocol [SOSConnectingViewController](#).

SOSUIPhaseScreenSharing

ScreenSharing Phase. This phase is the UI shown to the user during the screen sharing phase. The class associated with this phase must extend [SOSSessionBaseViewController](#) and implement the protocols [SOSUILineDrawingReceiveable](#) and [SOSUIAgentStreamReceiveable](#).

Declared In

`SOSOptions.h`

SOSFramework String Constants

List of string constants associated with the Service SDK SOS framework.

Table 14: String Constants

String Token	Description	Default Value
ServiceCloud.SOS.Title	Dialogs *//* The title we present for any UI element that has a title, such as one of our dialogs	Salesforce SOS
ServiceCloud.SOS.ConnectPrompt	The dialog body we present when prompting to start an SOS session	You are about to start an SOS session. You will be able to talk and screen share with an Agent in real time. Are you ready to experience the future of customer service?
ServiceCloud.SOS.Session.Disconnect	The dialog body we present when confirming that a user would like to end their session (after hitting an end session button)	Press OK to disconnect from the session.
ServiceCloud.SOS.End.Session.Remote	The dialog body we present when the agent has ended the SOS session	The Agent has ended the SOS session.
ServiceCloud.SOS.End.Session.AgentMissing	The dialog body we present when the agent disconnects for an unknown reason	The agent may have disconnected. Press Quit to end the session now or Cancel to continue waiting.
ServiceCloud.SOS.Continue.Waiting.Prompt	The dialog body we present when asking a customer if they would like to continue waiting for a session	All agents are currently busy, do you wish to continue waiting?
ServiceCloud.SOS.End.Session.Timeout	The dialog body we present when we cannot start a session due to no agents being online"	Your SOS Session has timed out. Please try again.

String Token	Description	Default Value
ServiceCloud.SOS.End.Session.InsufficientNetwork	The dialog body we present when we cannot start a session due to insufficient network conditions	Network not able to support SOS, please try again later.
ServiceCloud.SOS.Network.Test.Error	The dialog body we present if the network test encounters an error	We are unable to conduct a network test. Unfortunately we cannot guarantee an awesome SOS customer experience. Would you like to continue anyway?
ServiceCloud.SOS.End.Session.No.Agent	The dialog body we present if no agents are current available	No agents are currently available, please try again later.
ServiceCloud.SOS.End.Session.Network.Lost	The dialog body we present if the network no longer functions	Your session has ended because the network has become unavailable. Please try again.
ServiceCloud.SOS.Disconnect.BackgroundedWhileConnecting	The dialog body we present if the app was backgrounded before the connection process finished	Your session has ended because the app was backgrounded before the session was fully established. Please try again.
ServiceCloud.SOS.Disconnect.Reconnect.Failure	The dialog body we present is the reconnect of a session fails	Failed to re-establish to the SOS session. Please try again.
ServiceCloud.SOS.Connect.Positive	The positive choice for the start session prompt (saying yes to the dialog)	OK
ServiceCloud.SOS.Connect.Connect	A positive choice for connecting to a session. This is shown in a session prompt	Connect
ServiceCloud.SOS.Connect.Negative	The negative choice for the start session prompt (saying no to the dialog)	Cancel
ServiceCloud.SOS.Disconnect.EndNow	A button message in a prompt to end a session now	End Now
ServiceCloud.SOS.Continue.Waiting.Negative	The negative choice for the continue waiting prompt (saying no to the dialog)	Quit
ServiceCloud.SOS.Session.Is.Agent.Joining	The notification message used when the agent is in the process of joining the SOS session	Agent now joining...
ServiceCloud.SOS.Session.Is.Starting	The notification message used when a session is starting up	Session is starting...
ServiceCloud.SOS.Session.Is.Initializing	The notification message used when a session is initializing	Initializing...
ServiceCloud.SOS.Session.Is.Waiting.For.Agent	The notification message used when the customer is ready to start a session, but still waiting for the agent to join	Waiting for an agent...
ServiceCloud.SOS.Session.Is.Reconnecting.Audio	The notification message used when the audio needs to be reconnected	Reconnecting Audio...

String Token	Description	Default Value
ServiceCloud.SOS.Network.Test.Is.Initializing	The notification message used the the network is initializing	Initializing Network...
ServiceCloud.SOS.Session.Is.Reconnecting.Network	The notification message used the a switching of networks has occurred and reconnect is required	Switching network, reconnecting...
ServiceCloud.SOS.UI.Status.Queued	None Blocking UI */ /* Status message displayed when the session is in the agent queue	In queue
ServiceCloud.SOS.UI.Status.Connecting	Status message displayed when the session is connecting to an agent	Connecting
ServiceCloud.SOS.UI.Status.Audio.Muted	Status message displayed when the user's audio is muted	Mic muted
ServiceCloud.SOS.UI.Status.Poor.Network	Status message displayed network connection is poor during a session	Poor network
ServiceCloud.SOS.Session.Is.Paused	Status message displayed when agent video is paused	Session paused
ServiceCloud.SOS.Agent.Is.Paused	Status message displayed when the agent pauses the session	Session on hold
ServiceCloud.SOS.Session.Is.Paused.Alt	Status message displayed as a second message when the video is paused	(agent cannot see screen)
ServiceCloud.SOS.Agent.Is.Paused.Alt	Status message displayed as a second message when the agent pauses the session	(agent has paused session)
ServiceCloud.SOS.Disconnect.Prompt	Status message displayed when the disconnect yes or no options are shown	Disconnect from the session?
ServiceCloud.SOS.All.Agent.Busy	Status message displayed when all the agents are busy timeout is shown	All Agents are currently busy: Continue waiting?
ServiceCloud.SOS.Recording	Agent View */ /* Text overlayed on the agent container when the SOS session is being recorded	recording
ServiceCloud.SOS.Network.Test.Fail	Message that informs the user a network test is not possible	Unable to complete network test. Proceed anyways?
ServiceCloud.SOS.Network.Test.UnableToConnect	Message that informs the user the session could not connect	Unable to connect... Try again?
ServiceCloud.SOS.Session.Agent.Joined	The notification message used when an agent has joined and the app is backgrounded	Agent now joining, click here to return to the app

String Token	Description	Default Value
ServiceCloud.SOS.Onboarding.Header.Label	OnBoarding *//* The header text shown at the top of the onboarding UI during session start.	SOS
ServiceCloud.SOS.Onboarding.Carousel.Intro.Text	The text displayed on the first default card of the onboarding dialog.	You are about to co-browse your app with an expert who can see and draw on your screen. Don't worry, they can't see you :)
ServiceCloud.SOS.Onboarding.Carousel.Dragging.Text	The text displayed on the second default card of the onboarding dialog.	Drag the SOS window anywhere on your screen or tap to reveal controls.
ServiceCloud.SOS.Onboarding.Carousel.Controls.Text	The text displayed on the third default card of the onboarding dialog.	Tap to reveal controls to mute mic, pause screen share, and end session.
ServiceCloud.SOS.Onboarding.AcceptButton.Label	The button text displayed on the onboarding UI's Accept button	OK, GOT IT
ServiceCloud.SOS.Video.Pause.Label	Session Controls Camera View *//* The label under the pause/resume button in the video activity when the session is NOT paused.	Pause
ServiceCloud.SOS.Video.Resume.Label	The label under the pause/resume button in the video activity when the session is PAUSED.	Resume
ServiceCloud.SOS.Video.Unmute.Label	The label under the mute/unmute button in the video activity when the session is MUTED.	Unmute
ServiceCloud.SOS.Video.Mute.Label	The label under the mute/unmute button in the video activity when the session is NOT muted.	Mute
ServiceCloud.SOS.Video.End.Label	The label under the end session button in the video activity.	End
ServiceCloud.SOS.Video.Gesture.Info.Camera.Swap	Camera View *//* Text displayed on the Video UI to describe the camera swap gesture.	Double tap to flip camera
ServiceCloud.SOS.Video.Gesture.Info.Flashlight	Text displayed on the Video UI to describe the flashlight gesture.	Tap & hold anywhere to toggle flashlight
ServiceCloud.SOS.Video.Gesture.Info.Dismiss	Text displayed on the dismissal button for gesture information on the Video UI.	GOT IT
ServiceCloud.SOS.Video.Back.To.App.Label	Header that appears in the video activity that describes to the user they can click the header to return to the app.	Back to app
ServiceCloud.SOS.Video.Flashlight.On.Label	Label that describes a flashlight icon. Indicates that the phone flashlight is on	Flashlight On

String Token	Description	Default Value
ServiceCloud.SOS.Video.Muted.Label	Message that informs the user that the device has been muted	Muted
ServiceCloud.SOS.Video.Session.Pause.Header	The header of the message shown to the user when the session is paused in camera mode	Session paused
ServiceCloud.SOS.Video.Agent.Pause.Header	The header of the message shown to the user when the agent is paused in camera mode	Session on hold
ServiceCloud.SOS.Video.Session.Pause.Info	The text that appears below the header whenever the session is paused in camera mode	No one can see or hear you
ServiceCloud.SOS.Video.Agent.Pause.Info	The text that appears below the header whenever the agent is paused in camera mode	Agent has paused session
ServiceCloud.SOS.Video.Connecting.Label	Message that informs the user that the camera view is connecting	Connecting...

Additional Resources

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

- **Service SDK:** More info about the Service SDK.
 - [Service SDK Landing Page](#)
- **Salesforce Mobile SDK:** The SDK that lets you build Salesforce applications for mobile devices.
 - [Mobile SDK Landing Page](#)
 - [Mobile SDK Developer's Guide](#)
 - [Mobile SDK Trailhead](#)
- [Salesforce Developer Documentation](#): Landing page for developer documentation at Salesforce.
- [Salesforce Help](#): Landing page for general documentation at Salesforce.

INDEX

A

- action button customization [144](#)
- activate cases interface [80](#)
- activate knowledge interface [61](#)
- additional resources [298](#)
- agent availability
 - Live Agent [102](#)
 - SOS [122](#)
- app store submission [32](#)
- assigning permissions in sos [21](#)
- authenticated knowledge [58](#)
- authentication [58](#), [77](#)
- auto case pop in sos [22](#)
- automated email responses [93](#)

B

- branding [131](#)

C

- caching [62](#)
- case deflection [87](#)
- case management [71–72](#), [77](#)
- case management cloud setup [11](#)
- case publisher [71–72](#)
- case publisher setup [73](#)
- CasesFramework String Constants [218](#)
- ChatFramework String Constants [237](#)
- check Live Agent availability [102](#)
- check SOS agent availability [122](#)
- color customization [132](#)
- community cloud setup [10](#)
- community url [7](#)
- configure live agent chat session [99](#)
- configure sos session [112](#)
- custom case field data [84](#)
- custom data in sos [127](#)
- customize sos cloud setup [21](#)
- customize success view [89](#)

D

- data category [7](#)
- data category group [7](#)
- disable screen sharing [124](#)
- dynamic libraries [32](#)

E

- enable case management from knowledge [70](#)
- errors in sos [117](#)
- events in sos [117](#)
- example app
 - SOS [106](#)

F

- field masking in sos [124](#)
- file transfer [103](#)
- font customization [137](#)
- fonts [145](#)

H

- hidden case fields [84](#)

I

- image customization [139](#)
- install sdk [29–30](#), [32](#)

K

- knowledge [51–52](#)
- knowledge cloud setup [7](#), [10](#)
- knowledge interface [61](#)
- knowledge setup [53](#)
- KnowledgeFramework String Constants [206](#)

L

- launch from web view [149](#)
- listen to sos events [117](#)
- live agent chat [93–94](#)
- live agent chat cloud setup [13](#)
- live agent chat setup [95](#)
- logging [153](#)

M

- mask field in sos [124](#)
- mask field programmatically [126](#)
- mask field with storyboard [125](#)
- multiple queues in sos [27](#)

N

- notifications for case activity [92](#)

O

offline access [62](#)

P

prerequisites [28](#)
 push notifications [147](#)
 push notifications for case activity [92](#)

Q

quick setup
 case publisher [73](#)
 knowledge [53](#)
 live agent chat [95](#)
 sos [107](#)
 quick start [33](#)

R

reference [160](#), [208](#), [220](#), [239](#)
 release notes [2](#)
 remote notifications [147](#)
 replace sos ui [128](#)
 resources [298](#)

S

SCAppearanceConfiguration Class Reference [161](#)
 SCAppearanceConfigurationDelegate Protocol Reference [166](#)
 SCArticleSortByField Constants Reference [180](#)
 SCArticleSortOrder Constants Reference [181](#)
 SCCaseInterface Class Reference [208](#)
 SCChannelType Constants Reference [182](#)
 SCKnowledgeInterface Class Reference [183](#)
 SCKnowledgeInterfaceDelegate Protocol Reference [184](#)
 SCQueryMethod Constants Reference [185](#)
 screen sharing [124](#)
 SCSArticle Class Reference [186](#)
 SCSArticleDownloadOption Constants Reference [189](#)
 SCSArticleQuery Class Reference [189](#)
 SCSArticleViewController Class Reference [192](#)
 SCSArticleViewControllerDelegate Protocol Reference [192](#)
 SCSCaseDetailViewController Class Reference [216](#)
 SCSCaseDetailViewControllerDelegate Protocol Reference [217](#)
 SCSCaseListViewViewController Class Reference [210](#)
 SCSCaseListViewViewControllerDelegate Protocol Reference [211](#)
 SCSCaseNotification Class Reference [218](#)
 SCSCasePublisherResult Constants Reference [212](#)
 SCSCasePublisherViewController Class Reference [213](#)
 SCSCasePublisherViewControllerDelegate Protocol Reference [213](#)
 SCSCategory Class Reference [195](#)
 SCSCategoryGroup Class Reference [197](#)

SCSChat Class Reference [221](#)
 SCSChatConfiguration Class Reference [224](#)
 SCSChatDelegate Protocol Reference [226](#)
 SCSChatEndReason Constants Reference [228](#)
 SCSChatErrorCode Constants Reference [229](#)
 SCSChatSessionState Constants Reference [230](#)
 SCSServiceCloud Class Reference [167](#)
 SCSServiceCloudDelegate Protocol Reference [172](#)
 SCSServiceErrorCode Constants Reference [175](#)
 SCSKnowledgeErrorCode Constants Reference [199](#)
 SCSKnowledgeManager Class Reference [199](#)
 SCSMutableArticleQuery Class Reference [204](#)
 SCSNotification Class Reference [175](#)
 SCSNotificationType Constants Reference [176](#)
 SCSPrechatObject Class Reference [231](#)
 SCSPrechatPickerObject Class Reference [232](#)
 SCSPrechatPickerOption Class Reference [234](#)
 SCSPrechatTextInputObject Class Reference [235](#)
 SCSServiceConfiguration Class Reference [176](#)
 sdk install [29–30](#), [32](#)
 SDK prerequisites [28](#)
 sdk setup [27](#)
 service cloud setup [6](#)
 service sdk developer's guide [1](#)
 ServiceCommon String Constants [179](#)
 session recording in sos [25](#)
 setup [6](#), [27](#)
 sos [103–104](#)
 sos cloud manual setup [20](#)
 sos cloud quick setup [17](#)
 sos cloud setup [15](#)
 sos console quick setup [17](#)
 SOS example app [106](#)
 sos reference id [26](#)
 sos setup [107](#)
 sos video [114](#)
 SOSAgentAvailability Class Reference [241](#)
 SOSAgentAvailabilityDelegate Protocol Reference [243](#)
 SOSAgentAvailabilityStatusType Constants Reference [244](#)
 SOSCameraType Constants Reference [244](#)
 SOSConnectingBaseViewController Class Reference [245](#)
 SOSConnectingViewController Protocol Reference [246](#)
 SOSDelegate Protocol Reference [247](#)
 SOSErrorCode Constants Reference [250](#)
 SOSFramework String Constants [285](#), [294](#)
 SOSLogger Class Reference [252](#)
 SOSLoggerProtocol Protocol Reference [253](#)
 SOSLogStream Constants Reference [254](#)
 SOSLogTimerCode Constants Reference [256](#)

SOSMaskable Protocol Reference [257](#)
SOSMaskedTextField Class Reference [257](#)
SOSMasking Class Reference [259](#)
SOSMaskState Constants Reference [260](#)
SOSNetworkStatus Constants Reference [261](#)
SOSOnboardingBaseViewController Class Reference [262](#)
SOSOnboardingViewController Protocol Reference [263](#)
SOSOptions [112](#)
SOSOptions Class Reference [264](#)
SOSReconnectStatus Constants Reference [269](#)
SOSScreenAnnotations Class Reference [270](#)
SOSScreenSharing Class Reference [271](#)
SOSScreenSharingBaseViewController Class Reference [272](#)
SOSSessionBaseViewController Class Reference [273](#)
SOSSessionManager Class Reference [277](#)
SOSSessionState Constants Reference [282](#)
SOSSessionViewController Protocol Reference [283](#)
SOSStopReason Constants Reference [284](#)
SOSTelemetryLogStream Constants Reference [289](#)
SOSUIAgentStreamReceivable Protocol Reference [290](#)
SOSUILineDrawingReceivable Protocol Reference [292](#)
SOSUIPhase Constants Reference [293](#)
stock images [139](#)

T

transfer file [103](#)

troubleshooting
 app store [156](#)
 community [154](#)
 network [157](#)
tutorial
 case publisher [37](#)
 knowledge [33](#)
 live agent chat [41](#)
 sos [47](#)
two-way video [114–115](#)

U

ui customization
 action buttons [144](#)
 colors [132](#)
 fonts [137](#), [145](#)
 images [139](#)
UIWebView [149](#)
using case management [71–72](#)
using case publisher [71–72](#)
using knowledge [51–52](#)
using live agent chat [93–94](#)
using sos [103–104](#)

W

web view [149](#)
web-to-case [93](#)