Service Cloud Voice for Partner Telephony Developer Guide

Version 64.0, Summer '25

Last updated: June 20, 2025

CONTENTS

CHAPTER 1 Service Cloud Voice for Partner Telephony Developer Guide

Connect your telephony system with Service Cloud Voice, creating a unified and intuitive rep experience to give customers faster and more personalized service. Salesforce process automation offers recommendations, initiates workflows, and reduces post-call handle time to help reps resolve calls faster. If you're a telephony provider that wants to integrate your system with Voice, this guide is for you. We'll walk you through creating a connector and package for customers so they can add the power of Voice to your own system.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

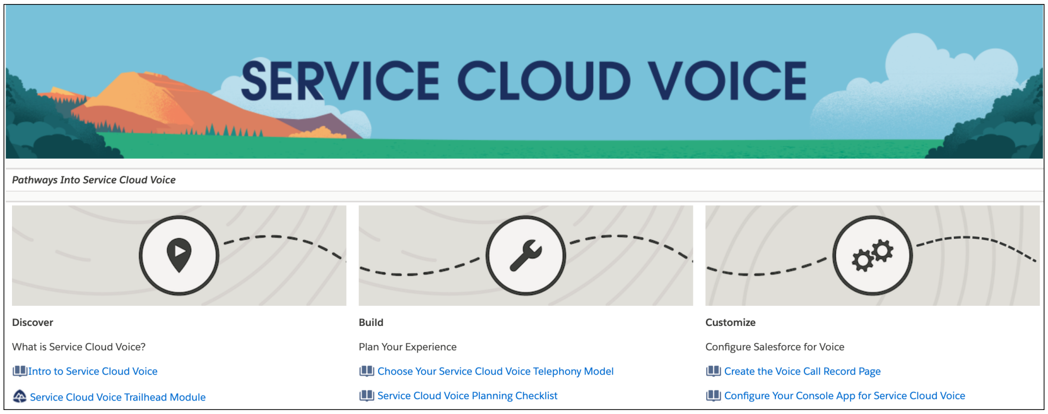
Be sure to bookmark the Service Cloud Voice Learning Map, a centralized collection of Voice resources that provides documentation, demos, and more for every step in your Voice implementation.

EDITIONS

Available in: Lightning Experience

Available in: **Performance**, **Enterprise**, **Unlimited**, and **Developer** Editions

Available in: Sales Cloud, Service Cloud, and Government Cloud as an add-on license. Government Cloud is supported only on Service Cloud Voice with Amazon Connect and Service Cloud Voice for Partner Telephony.



SERVICE CLOUD VOICE

Pathways Into Service Cloud Voice

Discover

What is Service Cloud Voice?

Intro to Service Cloud Voice

Service Cloud Voice Trailhead Module

Build

Plan Your Experience

Choose Your Service Cloud Voice Telephony Model

Service Cloud Voice Planning Checklist

Customize

Configure Salesforce for Voice

Create the Voice Call Record Page

Configure Your Console App for Service Cloud Voice

SEE ALSO:

Service Cloud Voice Learning Map

Salesforce Help: Service Cloud Voice

Salesforce Help: Set Up Service Cloud Voice with Partner Telephony

Trailhead: Service Cloud Voice

CHAPTER 2 Recent Changes to the Service Cloud Voice for Partner Telephony Developer Guide

This section describes recent changes that have been made to this guide.

Updates for Summer ’25

- Updated Unified Routing (Beta) on page 101 because unified routing now supports outbound calls, skills-based routing, and call transfers.

Updates for Spring ’25

- Moved the Service Cloud Contact Center Connector API reference to a standalone reference guide and removed Contact Center from the title. Access the Service Cloud Connector API Reference in its new location.
- Added Disable Call Actions on page 112, and related field updates for VoiceCapabilitiesResult, CallInfo, and publishEvent.
- Added Headset Support on page 117 which allows reps to control call handling actions using headset, and related field updates for VoiceCapabilitiesResult.
- Updated Desk Phone Support on page 113 which allows reps to use a desk phone even if microphone settings are disabled in the browser. With this change, we store the `selectedPhoneType` value and for this release, the `param.selectedPhone` field of the `AgentConfigResult` object will be reset to default value due to this change. Added SetAgentConfigResult object.
- Added Unified Routing (Beta) on page 101 so that Salesforce handles the routing of the inbound voice calls in the voice channels.

Updates for Winter ’25

- Updated the event type and error type (`eventType`) values in various code samples and pages of the Connector API Reference to reflect backend changes. For example, changed VOICE.CALL_CONNECTED to VOICE_EVENT_TYPE.CALL_CONNECTED and changed LOGIN_REQUIRED to SHARED_ERROR_TYPE.LOGIN_REQUIRED.
- Added the `customIcon` field in the Conversation Channel Definition record in Create a Salesforce DX Project on page 11. This new field helps admins identify the contact center partner during setup.
- Added the `getContacts` method to the Connector API Reference under the Common API Methods section. The `getContacts` method returns a new `ContactsResult` object.
- Reorganized capabilities into shared and voice categories, resulting in these changes:

  - Removed support for the `CapabilitiesResult` object in the Connector API Reference. Use the `SharedCapabilitiesResult` object or `VoiceCapabilitiesResult` object instead.
  - Removed support for the `getCapabilities` method in the Connector API Reference. Use the `getSharedCapabilities` method or `getVoiceCapabilities` method instead.

Updates for Summer ’24

- Added instructions for how to Let Reps Control the Callback Experience on page 62.

Updates for Spring ’24

- Moved the `ConversationVendorInfo` topic to the Object Reference for the Salesforce Platform Guide.
- Updated the Connector API Reference with the following changes: separated the Connector API Methods into a Common API Methods section and a Voice API Methods section; and updated the event type (`eventType`) values in various code samples and pages to reflect backend changes, for example, changed CALL_CONNECTED to VOICE.CALL_CONNECTED.
- Added the `getTelephonyConnector` method to the Connector API Reference. The `getTelephonyConnector` method returns an object of the TelephonyConnector class.
- Added the `onAgentWork` method to the Connector API Reference. The `onAgentWork` method sends non-Voice related AgentWork events to the partner contact center
- Added the `AgentWork` object to the Connector API Reference. The `AgentWork` object stores information about an AgentWork record.

Updates for Winter ’24

- Added the ability to Set the Voice Call Record Type on page 65 to determine the page layout of a voice call record.
- Added a `disconnectReason` parameter to the Update a Voice Call Record Telephony Integration API to pass in the reason why a voice call was disconnected.
- Added the ability to Send Real-Time Conversation Events on page 81 generated from intelligence sources to the rep console. Also added Apex classes and interfaces to the Apex Reference on page 125 section to support real-time signals.

Updates for Summer ’23

- Added the Create Transcripts in Bulk Telephony Integration API to support bulk transcriptions in real time.
- Added steps on how to Customize Error Messages. Also added the CustomError Connector API object to support the feature.

- Added steps on how to Enable Agent-to-Agent Calls. Also added the `hasPhoneBook` parameter to the CapabilitiesResult Connector API object.
- Added the Service Cloud Voice Lightning Web Component Toolkit API Telephony Actions page to support LWC telephony actions.
- Added the `addParticipant`, `endCall`, `merge`, `sendDigits`, and `swap` Toolkit API telephony actions to automate softphone call control events. The actions are supported for both Lightning Web Component and Aura components.
- Updated the Create a Voice Call Record Telephony Integration API to state that `callAttributes` now supports call transfers and callbacks.

Updates for Spring ’23

- Added more guidelines regarding call transcription: Transcribe Calls in Real Time.
- Added Hide Call Controls and the associated new show methods in CallInfo.
- Added `UpdateOrgDomainProvider` to the list of supported Apex interfaces. Implement this interface to get notified of My Domain changes in your org, which require updates to Service Cloud Voice. See Service Cloud Voice for Partner Telephony Apex Reference.
- Added steps on how to Configure Estimated Wait Times for Queues on page 106 and associated fields to the CallInfo, CapabilitiesResult, and Contact objects.
- Added steps on how to Enable the Voice Extension Page in Lightning App Builder on page 97. Also introduced a VoiceExtension metadata type to support the Voice Extension FlexiPage.
- Added Associate Partner Telephony Users and Groups with Queues.

Updates for Winter ’23

- This guide has been reorganized so that it’s easier to find information.
- Updated information in the Outbound Dialers section about how to use a preview dialer. Related updates:
  - Added the `startPreviewCall` method to the Toolkit API Lightning component.
  - Added the `CallOrigin` field to the VoiceCall object (in the Object Reference Guide).
- New topic: Send Voicemails to Reps. Related updates:
  - Added a `callOrigin` parameter to Create a Voice Call Record Telephony Integration API.
  - Added a `callOrigin` parameter to Update a Voice Call Record Telephony Integration API.
- New topic: Transfer Calls to a Queue. Related updates:
  - Added a `queue` parameter to Create a Voice Call Record Telephony Integration API.
- New topic: Change Status While on a Call.

Updates for Summer ’22

New topics:

- Perform a Blind Transfer

- Use Click-to-Dial for Transfers
- Customize the Destination List for Call Transfers in Omni-Channel on page 107
- Enable Voice Call Transfers via Salesforce Omni-Channel Flows on page 108
- Connector & Supervisor API
  - CapabilitiesResult—Moved existing capabilities from AgentConfigResult to this new object. Added capabilities for new features (supervisor barge in, blind transfer, transfer to Omni-Channel flow, filter by transfer category).
  - getCapabilities
  - supervisorBargeIn
  - SupervisedCallInfo
  - hasTransferToOmniFlow on page 108

Changes to existing topics:

- Quick Start Partner Telephony Package on page 17. Added the ServiceCloudVoice component, which is automatically enabled when the quick-start-partner-telephony package is installed.
- Supervisor Listen In or Barge In on page 84—Added info about Supervisor Barge In and changed all references from AgentConfigResult to the new CapabilitiesResult.
- Troubleshooting—Added info about the possible causes when the Omni-Channel widget doesn't show AgentWork.
- Connector & Supervisor API
  - AgentConfigResult—Moved some of the params to CapabilitiesResult.
  - getPhoneContacts—Added a mention to contact types (not just contacts).
  - PhoneContactsResults—Added a parameter for contact types.
  - publishEvent—Added new events for Supervisor Barge In.
  - SupervisorEvents—Added new events for Supervisor Barge In.

Updates for Spring ’22

New topics:

- Add Contact Center Channels to Enable Routing on page 95
- Add a Partner Settings UI to Omni-Channel on page 47
- Two-Way Rep Status Syncing on page 98
- Automated Key Provisioning on page 47
- Supervisor Listen In on page 84
- Outbound Dialers on page 63
- Line-Specific Controls on page 123
- Supervisor API
- Universal Call Recording Playback in Call Recordings on page 88

Changes to existing topics:

- Apex Reference on page 125—Added KeyProvider and PhoneNumberProvider.

- ConversationVendorInfo—Added keyProvisioningSupported, partnerPhoneNumberSupported, and universalCallRecordingAccessSupported.
- Connector API
  - publishEvent—Added supervisor events.
  - AgentStatusInfo—New class.
  - CallInfo—New class.
- Troubleshooting on page 131—Added a Download Connector Logs section.

CHAPTER 3 Set Up and Publish Your Partner Telephony Package

In this chapter ...

To get started with Service Cloud Voice for Partner Telephony, set up and publish a managed package. This package lets you develop and distribute resources that are needed to integrate your telephony system with Service Cloud Voice and enable Service Cloud Voice in an org.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Note: We assume that you're a Salesforce admin who is comfortable using git, generating public and private keys, and working with Salesforce APIs.

To integrate your telephony system with Service Cloud Voice, you'll also include a connector in your package that lets Salesforce and your telephony system talk to each other. The connector enables inbound and outbound calling as well as common call controls such as mute, hold, and transfer.

A Note About Scratch Orgs

When developing your managed package, we recommend using a scratch org. The scratch org is a source-driven and disposable deployment of Salesforce code and metadata.

A scratch org is fully configurable, so developers can emulate different Salesforce editions with different features and preferences. You can share the scratch org configuration file with other team members so you all have the same basic org in which to do your development.

The scratch org definition file contains the configuration values that determine the shape of the scratch org. To enable Service Cloud Voice Partner Telephony features in a scratch org, specify “ServiceCloudVoicePartnerTelephony” in the features field in your scratch org definition. Add a quantity value (between 1–50) when you add the ServiceCloudVoicePartnerTelephony scratch org feature. To learn more about all of the supported features, see Scratch Org Features.

Note: To learn more about scratch orgs and the Salesforce Developer Experience (DX), see How Salesforce Developer Experience Changes the Way You Work.

SEE ALSO:

Salesforce DX Developer Guide: Scratch Org Features

Create a Service Cloud Voice Package

This managed package lets you develop and distribute resources that are needed to integrate your telephony system with Service Cloud Voice and enable Voice in an org.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that’s not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

1. Create Developer Hub and Namespace Orgs

   Linking a Developer Hub to a namespace org lets you use the Salesforce Developer Experience to develop a second-generation package.

2. Create a Salesforce DX Project

   Allow Service Cloud Voice to communicate with your telephony provider. The package you’re creating includes a connector, contact center, settings metadata, and more.

3. Create and Deploy Your Package

   Use these commands to create, update, and install the package.

4. Assign User Permissions

   Service Cloud Voice for Partner Telephony comes with the following user permissions, which should be assigned to users as part of the Contact Center Admin (Partner Telephony) and Contact Center Agent (Partner Telephony) permission sets.

5. Generate a Self-Signed Certificate with OpenSSL

   Use OpenSSL to generate an RSA private key and certificate.

Create Developer Hub and Namespace Orgs

Linking a Developer Hub to a namespace org lets you use the Salesforce Developer Experience to develop a second-generation package.

1. Create an org with Dev Hub features enabled.
   a. Create a Developer Edition org.
   b. Enable Lightning Experience.
   c. Enable Dev Hub features: From Setup, enter *Dev Hub* in the Quick Find box and select **Dev Hub**. Then, click **Enable**. After you enable Dev Hub, you can't disable it.
   d. Enable Unlocked Packages and Second-Generation Managed Packages.

2. Create a namespace org.
   a. Create a Developer Edition org.
   b. Enable Lightning Experience.
   c. Create a namespace.

      Note: Choose your namespace carefully. If you're trying out this feature or need a namespace for testing purposes, choose a disposable namespace. Don't choose a namespace that you want to use in the future for a production org or some other real use case. After you associate a namespace with an org, you can't change it or reuse it.

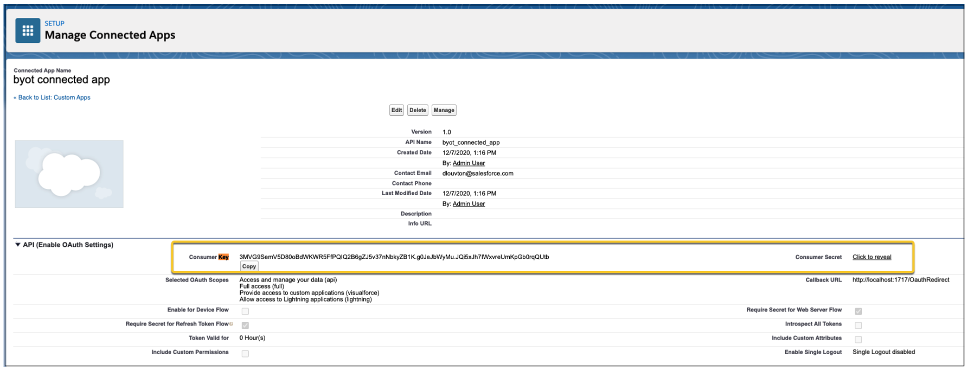
   d. In the Dev Hub org, use Namespace Registry to register the namespace that you created. To learn more, see Link a Namespace to a Dev Hub Org.
   e. In the `sfdx-project.json` file, specify your namespace using the **namespace** attribute.

3. Create a connected app in your Dev Hub org for authorization. The app allows you to set the refresh token timeout, specify IP ranges, and more. Use any name; for example, Dev Hub Connected App.

4. To authenticate into the dev hub, use the following Salesforce CLI command. For *HUB_ORG_ALIAS*, choose any name. For *CLIENT_ID*, use the consumer key. For *INSTANCE_URL*, use https://login.salesforce.com.

```
sf org login web --set-default-dev-hub --alias [HUB_ORG_ALIAS] --client-id [CLIENT_ID]
 —instance-url [INSTANCE_URL]
```

5. You are prompted for the secret. To retrieve it, click **Click to reveal** under Consumer Secret.



SETUP
Manage Connected Apps

Connected App Name
byot connected app

« Back to List: Custom Apps

Edit Delete Manage

Version 1.0
API Name byot_connected_app
Created Date 12/7/2020, 1:16 PM
By: Admin User
Contact Email dlouvton@salesforce.com
Contact Phone
Last Modified Date 12/7/2020, 1:16 PM
By: Admin User
Description
Info URL

▼ API (Enable OAuth Settings)

Consumer Key 3MVG9SemV5D80oBdWKWR5FfPQIQ2B6gZJ5v37nNbkyZB1K.g0JeJbWyMu.JQi5xJh7IWxvreUmKpGb0rqQUtb Copy
Consumer Secret Click to reveal
Selected OAuth Scopes Access and manage your data (api)
Full access (full)
Provide access to custom applications (visualforce)
Allow access to Lightning applications (lightning)
Callback URL http://localhost:1717/OauthRedirect
Enable for Device Flow □
Require Secret for Web Server Flow ☑
Require Secret for Refresh Token Flow ☑
Introspect All Tokens □
Token Valid for 0 Hour(s)
Include Custom Attributes □
Include Custom Permissions □
Enable Single Logout Single Logout disabled

Tip: To view the OAuth client ID and personal connected app secret, from Setup, enter *App Manager* in the Quick Find box and click **App Manager**. Click **Connected App** **Consumer Key** and **Consumer Secret**. To learn about more authentication methods, see Authorize an Org Using the Web Server Flow.

Create a Salesforce DX Project

Allow Service Cloud Voice to communicate with your telephony provider. The package you're creating includes a connector, contact center, settings metadata, and more.

1. Clone the `scv-partner-telephony-quickstart` GitHub repository, and install Salesforce CLI.

   ```
   git clone --recurse-submodules
   https://github.com/service-cloud-voice/scv-external-telephony-quickstart
   cd scv-external-telephony-quickstart
   npm install # This will install Salesforce CLI
   ```

   - You can also install the Salesforce CLI separately. To learn more, see Salesforce CLI.
   - If you already have Salesforce CLI installed, make sure to update it to the latest version `sf update`. This step is important because we've added our new Metadata type support.
   - Make sure to use API version 53.0 in the Salesforce DX project file `sfdx-project.json`. If you have the local API version override, make sure it's set to 53.0.
   - If you aren't using API version 53.0 or are using an older Salesforce CLI version, you may see errors related to ConversationVendorInfo Metadata type. The error message from Salesforce CLI looks like this:

     ```
     Unexpected file found in package directory:
     /Users/.../scv-partner-telephony-quickstart/force-app/main/default/ConversationVendorInformation/sampleVendor.ConversationVendorInformation-meta.xml
     ```

2. Develop your connector. For a sample implementation, see the open-source demo connector repo in GitHub.
3. Update `ConversationVendorInfo` in `scv-partner-telephony-quickstart/force-app/main/default/ConversationVendorInformation/sampleVendor.ConversationVendorInformation-meta.xml`.

   Use the following guidelines when updating the fields in this file.

Field	Description
bridgeComponent	Set the bridgeComponent to: `<package-namespace>:<lms bridge name>`. If you aren't using it, you can remove it. If you're testing with the demo connector, you can use `{your_namespace}:lwcBridge` or `{your_namespace}:AuraBridge`. To learn more see Use the Lightning Message Service Bridge on page 52.
einsteinConversationInsightsSupported	Set this capability to `true` to support Einstein Conversation Insights for customers. Implement the `service_cloud_voice.RecordingMediaProvider` on page 125 interface in your Apex integration class to provide Salesforce call recording URLs for analysis. **NOTE:** To use this capability, you must also set `namedCredentialSupported` to `true`.
partnerContactCenterListSupported	Set this capability to `true` if the customer account has multiple contact centers and the customer has to select one to connect with Salesforce. **NOTE:** To use this capability, you must also set `namedCredentialSupported` to `true`.

Field	Description
namedCredentialSupported	Set this capability to `true` if you support prescriptive setup through a named credential. Implement the `service_cloud_voice.PartnerConnector` on page 125 interface in your Apex integration class, which tests the connection to the new customer account using the customer named credential
partnerTransferDestinationsSupported	Set this capability to `true` so that Salesforce can fetch rep queues in order for Salesforce and rep queues to be mapped. Implement the `service_cloud_voice.TransferDestinationProvider` on page 125 interface in your Apex integration class. **NOTE:** To use this capability, you must also set `namedCredentialSupported` to `true`.
agentSSOSupported	Set this capability to `true` if you support rep SSO using Salesforce as the identity provider. Implement the `service_cloud_voice.PartnerSSO` on page 125 interface in your Apex integration class. **NOTE:** To use this capability, you must also set `namedCredentialSupported` to `true`.
clientAuthMode	This field is deprecated in API version 53.0 and later. Possible values are “SSO”, “Custom” and “Mixed”.
connectorUrl	URL of the connector. **a.** If you’re testing with the demo connector, use the absolute link from the steps you performed in the `Use the Demo Connector` on page 32 page. For example, `https://www.myTelephonyDemo.com:8080`. **b.** You can also choose to host the connector as a Visualforce page from the same Salesforce managed package. Use the relative URL format from Host the Connector as a Visualforce Page on page 120. For example, `/apex/<namespace>__<connector visual force page name>`.
customConfig	The name of the custom object that defines the fields for storing custom settings for contact centers.
customIconId	ID of the static resource used to identify the contact center integration, such as a partner telephony provider logo. Add a custom icon to Salesforce by defining the static resource, and then use the ID value for this field. The static resource must be in SVG format. This field is optional.

Field	Description
customLoginUrl	The URL that is used to load the telephony system login page. If you're testing with the demo connector, use the link from the steps you performed in the `Use the Demo Connector` on page 32 page. For example, `https://www.myTelephonyDemo.com:8080/login.html`.
developerName	Set this value to a unique name. It must match the file name.
integrationClassName	This field is deprecated in API version 53.0 and later.
integrationClass	Set this foreign key to the *name* of the Apex class that implements methods to communicate to the vendor during the partner telephony setup. **NOTE**: If you've turned on any of the previous capabilities, you must provide the Apex class.
masterLabel	Set this to the desired display name for the telephony provider. This label shows up in the contact center record as the telephony provider.
namedCredential	Set this foreign key to the *name* of the named credential. You can choose to use this named credential or create one similar to this one. In both cases you have to update the named credential with the account secret credentials (for JWT, AWS Signature v4) or authorize the named credential (for OAuth-based named credentials).
serverAuthMode	This field is deprecated in API version 53.0 and later. Set to the serverAuthMode. Possible values are OAuth and None.
userSyncingSupported	Set this capability to `true` if you support automated user syncing whenever a user is added or removed to a contact center. Implement the `service_cloud_voice.UserSyncing` on page 125 interface in your Apex integration class, which is required for user syncing. **NOTE:** To use this capability, you must also set `namedCredentialSupported` to `true`.
vendorType	Set this value to `ServiceCloudVoicePartner`.

4. Update the following resources as needed. For details, see Use the Lightning Message Service Bridge on page 52. If you don't plan to use the Lightning Message Bridge, remove the `/aura`, `/lwc` and `/messageChannels` folders.

   a. **Aura Bridge:** This example is a sample Aura bridge component. When referring to the LMS channel, use the namespace as a prefix: `<namespace>__<lms channel name>__c`.

      i. In force-app/main/default/aura/AuraBridge/AuraBridge.cmp, change `<lightning:messageChannel type="REPLACE_WITH_NAMESPACE_NAME__ServiceCloudVoiceMessageChannel__c" ..` to `<lightning:messageChannel type="<namespace>__<lms channel name>__c" ..`

   b. **Aura LMS Sample:** This sample Aura component subscribes to the lightning message channel. When referring to the LMS channel, use the namespace as a prefix: `<namespace>__<lms channel name>__c`.

i. In force-app/main/default/aura/AuraLmsSample/AuraLmsSample.cmp, change `<lightning:messageChannel type="REPLACE_WITH_NAMESPACE_NAME__ServiceCloudVoiceMessageChannel__c" ..` to `<lightning:messageChannel type="<namespace>__<lms channel name>__c"`

c. **LWC Bridge:** This example is a sample LWC bridge component. When referring to the LMS channel, don't use the namespace as a prefix.

d. **LWC LMS Sample:** This is a sample LWC component that subscribes to the lightning message channel. When referring to the LMS channel, don't use the namespace as a prefix.

5. Other information about the sample resources that you can modify as necessary.

- The config custom object defines the fields for storing the custom settings of a contact center. A sample custom config is provided at `force-app/main/default/sampleCustomConfig__c`. This custom config, along with the associated layout, is used during contact center setup to fill out custom contact center details.
- A layout can be assigned to a custom object to specify how the custom object should be displayed. A sample layout is provided at `force-app/main/default/layouts`. This layout is referenced by `sampleCustomConfig__c` object.
- A named credential can be added to the package to be used by Apex callouts. A named credential specifies the URL of a callout endpoint and its required authentication parameters in one definition. Salesforce manages all authentication for Apex callouts that specify a named credential as the callout endpoint. A sample named credential is provided at `force-app/main/default/namedCredentials`.
- Lightning pages can be used to create a customized page with different Lightning components. A sample Lightning page is provided at `force-app/main/default/flexipage`.
- A utility bar is a specialized Lightning page that can provide quick access to commonly used tools. A sample utility bar for Service Cloud Voice is provided at `force-app/main/default/flexipage`.
- A Salesforce application can be used to group tabs and objects used for a specific functionality. A sample application for Service Cloud Voice is provided at `force-app/main/default/applications`.
- An Apex class on page 125 should be provided to automate setup operations that require server-to-vendor communication (for example, SSO, user syncing). The class must implement the interfaces corresponding to the capabilities that are supported as defined in the `ConversationVendorInfo`. A sample Apex integration class is provided at `force-app/main/default/classes`. Tests for the Apex classes should also be added so that the package can be published in the App Exchange.
- You can replace the placeholder namespace with your desired namespace in the SFDX project by running the following commands:

```
export LANG=C
export LC_CTYPE=C
find ./ -type f -exec sed -i '' 's/REPLACE_WITH_NAMESPACE_NAME'/'"<YOUR NAMESPACE
NAME>"'/g' {} \;
```

SEE ALSO:

GitHub: https://github.com/salesforce/scv-partner-telephony-quickstart

Create and Deploy Your Package

Use these commands to create, update, and install the package.

Create the Package

```
sf package create --name "<Package Name>" --path force-app --package-type Managed
--error-notification-username <Dev Hub Username>
```

Create a Package Version

```
sf package version create --package "<Package Name>" --code-coverage
--installation-key-bypass --wait 20
```

The same command is used to create newer versions of the package. This command generates an installation link that can be used in customer orgs.

```
Successfully created the package version [08cB0000000****IA0]. Subscriber Package Version
 Id: 04tB0000000d****IAQ
Package Installation URL:
https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB00000****DIAQ
(https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000d2wDIAQ)
```

Release a Package Version

Each new package version is marked as beta when created. As you develop your package, you may create several package versions before you create a version that is ready to be released and distributed. Only released package versions can be listed on AppExchange and installed in customer orgs.

Before you promote the package version, ensure that the **Promote a package version to released** user permission is enabled in the Dev Hub org associated with the package. Consider creating a permission set with this user permission, and then assigning the permission set to the appropriate user profiles.

When you're ready to release, use `sf package version promote --package "Expense Manager@1.3.0-7"`

If the command is successful, a confirmation message appears. `Successfully promoted the package version, ID: 04tB0000000719qIAA to released.`

After the update succeeds, view the package details. `sf package version report --package "Expense Manager@1.3.0.7"`

Confirm that the value of the Released property is *`true`*.

```
=== Package Version
NAME VALUE
────────────────────────────── ───────────────────

Name ver 1.0
Alias Expense Manager-1.0.0.5
Package Version Id 05iB0000000CaahIAC
Package Id 0HoB0000000CabmKAC
Subscriber Package Version Id 04tB0000000NPbBIAW
Version 1.0.0.5
Description update version
Branch
Tag git commit id 08dcfsdf
Released true
Created Date 2018-05-08 09:48
Installation URL
https://login.salesforce.com/packaging/installPackage.apexp?p0=04tB0000000NPbBIAW
```

You can promote and release only one time for each package version number, and you can't undo this change.

To learn more, see Workflow for Second-Generation Managed Packages.

Create a Scratch Org

You can create a scratch org to test your package. This process may take a few minutes. `sf org create scratch --definition-file config/project-scratch-def.json --target-org <Dev Hub Username>`

Open the Scratch Org

To find a list of scratch orgs, including the one you created, run this command.

```
sf org ist --verbose
```

To open the scratch org, run this command.

```
sf org open -u <scratch org username>
```

Install the Package

Before installing, make sure that the org has the **Service Cloud Voice Partner Telephony** license. Then, run this command:

```
sf package install --package "<Package Name>@<Package Version>" --target-org <Target
Org Username>
```

Target Org is the org where you want to install the package.

Or, use the installation URL that's created when you run the Salesforce CLI command for creating or promoting a package version.

Look for an email indicating whether the package was installed. If the installation failed, review the email for details and try again. To learn more about installation methods, see Use the CLI to Install a Second-Generation Managed Package.

Assign User Permissions

Service Cloud Voice for Partner Telephony comes with the following user permissions, which should be assigned to users as part of the Contact Center Admin (Partner Telephony) and Contact Center Agent (Partner Telephony) permission sets.

- Contact Center Agent for External Telephony Provider: Access contact centers that use an external telephony provider.
- Contact Center Admin for External Telephony Provider: Manage contact centers that use an external telephony provider.
- View Call Recording: View call recordings.
- Control Call Recording: Pause and resume recording of individual calls.

Generate a Self-Signed Certificate with OpenSSL

Use OpenSSL to generate an RSA private key and certificate.

You need the certificate to set up the contact center. The **reqTelephonyIntegrationCertificate** value should be the value in server.crt from these steps.

Important: **This key pair must be unique for each org installation.** Shared key pairs would allow the private key holder to call SCV APIs for any org using the corresponding public key. Subscriber orgs should be able to rotate this key on demand.

1. Create a folder to hold the generated certificate: `$ mkdir certificates`
2. Change the current directory to the certificates folder: `$ cd certificates`

3. In the certificates folder, specify a password and generate an RSA private key. Replace *<your_password>* with your own password.

   Note: The Certificate Authorities use this password to authenticate the certificate owner when they want to revoke their certificate. Because the certificate is self-signed, you can't revoke it using CRL (Certificate Revocation List).

   ```
   $ openssl genrsa -des3 -passout pass:<your_password> -out server.pass.key 2048
   ```

4. Create a key file from the server.pass.key file, using the password that you just created. `$ openssl rsa -passin pass:<your_password> -in server.pass.key -out server.key`
5. Delete the server.pass.key: `$ rm server.pass.key`
6. Request and generate the certificate: `$ openssl req -new -key server.key -out server.csr`
7. Enter the requested information. Press **Enter** when prompted for the challenge password. To skip entering a company name, enter a period `(.)`.
8. Generate the SSL certificate: `$ openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt`

Get Started with the Quick Start Partner Telephony Package

You can explore the partner telephony features in your partner telephony-enabled org. We've published a sample quick start package that you can install in your org, create a contact center, and get started.

The `quick-start-partner-telephony` package has a ConversationVendorInfo record. You can reference it while creating a contact center. It contains a connector URL which points to a Visualforce page (`quickStartPT__quickStartPTVFConnector`). You can change it to localhost (127.0.0.1). Both options require you to run a local server. For localhost, you must host the connector from `http://127.0.0.1`. For the Visualforce page, you need the server to call SCRT APIs.

The package also includes an LMS channel and Lightning web component/Aura record home and bridge components. The Visualforce page connector uses the AuraBridgeComponent. The localhost connector uses the lwcBridge component.

Installing the `quick-start-partner-telephony` package automatically enables MyDomain, Omni-Channel, and the external Service Cloud Voice telephony system.

The package includes these components:

Name	Type	Description
AuraLmsSample	Aura Component Bundle	Voice Call Record Home Aura component to demonstrate LMS
AuraMessageBridge	Aura Component Bundle	Aura bridge component needed for LMS messages
ServiceCloudVoiceMessageChannel	Lightning Message Channel	LMS channel
connectorPage	Visualforce Page	Connector page used as the connector URL
demoConnector	Static Resource	JS script referenced in the Connector VF page
loginPage	Visualforce Page	Demo login page used in Omni-Channel for partner login
login_page	Static Resource	JS script referenced in the login VF page

logo	Static Resource	Image icon
lwcBridge	Lightning Web Component Bundle	LWC bridge component needed for LMS messages
lwcLmsSample	Lightning Web Component Bundle	Voice Call record home Lightning web component (LWC) to demonstrate LMS
quickStartPTVFConnector	Conversation Vendor Information	ConversationVendorInfo record for VF page connector
quickStartPartnerTelephony	Conversation Vendor Information	ConversationVendorInfo record for localhost connector
remote_control	Static Resource	VF page used for simulator page (also known as "remote")
simulatorPage	Visualforce Page	JS script referenced in simulator VF page
slds_stylesheet	Static Resource	CSS for simulator VF page
symbols	Static Resource	Image icons
Service_Cloud_Voice (applications)	Application	Application containing components used for Service Cloud Voice
SampleIntegrationImpl.cls	Apex Class	Class that implements interfaces for automated setup
SampleIntegrationImplTest.cls	Apex Class	Tests for Apex integration class
Service_Cloud_Voice (FlexiPage)	FlexiPage	Lightning page for customized Service Cloud Voice page
Service_Cloud_Voice_UtilityBar	FlexiPage	Utility base with common tools for Service Cloud Voice
sampleCustomConfig__c-sampleCustomConfig Layout	Layout	Layout to define display for custom config object
sampleNamedCredential	Named Credential	Named credential to authenticate server-to-server communication for automated setup
sampleCustomConfig__c	Custom Object	Custom object for defining custom settings for contact centers
VoiceCall	Voice Call	Voice Call object
MyDomain	Setting	Verifies that My Domain is enabled
OmniChannel	Setting	Turns on Omni-Channel in the org
ServiceCloudVoice	Setting	Turns on the external Service Cloud Voice telephony system. If you've already installed the `quick-start-partner-telephony`

package, you can turn on the external Service Cloud Voice telephony system by copying the `ServiceCloudVoice.settings-meta.xml` file from the quick-start-partner-telephony package and adding it to the `force-app/main/default/settings` folder.

1. Install the quick-start-partner-telephony package in your org.
2. After installing the `quick-start-partner-telephony` package, follow the Partner Telephony Setup instructions to create a contact center and assign permission sets.
3. Import a contact center XML (see the sample XML in Set Up Service Cloud Voice for Partner Telephony in Your Org on page 21).
4. Set `reqVendorInfoApiName` as needed.
   - For the connector from localhost:

     ```
     <item sortOrder="2"
             name="reqVendorInfoApiName" label="Conversation Vendor Info Developer
             Name">quickStartPT__quickStartPartnerTelephony</item>
     ```

     quickStartPartnerTelephony is the developer name of the ConversationVendorInfo record where the connector URL is pointing to localhost (that is, `https://127.0.0.1:8080`). Make sure the connector is running on localhost 8080.

   - For the connector from the Visualforce page:

     ```
     <item sortOrder="2"
             name="reqVendorInfoApiName" label="Conversation Vendor Info Developer
               Name">quickStartPT__quickStartPTVFConnector</item>
     ```

     Start a server for SCRT server calls (such as inbound call, transcription, and call recording) and one initial call to set org details. The VF page included in the package uses localhost:3030 for SCRT calls (that is, `http://127.0.0.1:3030/`). Configure the org details using the configureTenantInfo call. Enable Cors on the server to enable calls to the other domain from the VF connector.

     - Run `npm install cors` in the demo connector.
     - Add the following two lines in `server.js`.

       ```
       import cors from 'cors';
       app.use(cors());
       ```

5. Set `reqVendorInfoApiName` as needed.

CHAPTER 4 Set Up Your Production Org

In this chapter ...

After you've installed the managed package, set up your production org.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Set Up Omni-Channel and a Lightning Console App

Service Cloud Voice uses Omni-Channel to send calls to reps in your Lightning console app. Reps use the Phone tab in the Omni-Channel utility to accept or decline calls and use other call controls.

❗ Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

To get started, assign the automatically generated custom permission, which contains the service presence status set to reps. Then create a Lightning console app and add the Omni-Channel utility to the app.

1. Assign Partner Telephony Permission Set to reps.

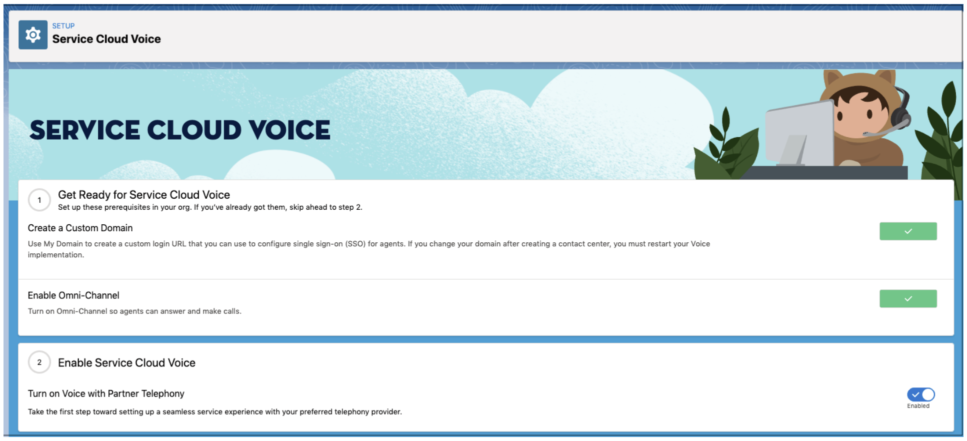
   There's an automatically generated permission set with the label "Partner Telephony Permission Set." This permission set contains an automatically generated Service Presence Status called "Available for Voice." Assign the reps in the contact center to this permission set.

2. Create a Lightning console app.
   a. From Setup in Lightning Experience, enter `App Manager` in the Quick Find box, then select **App Manager**.
   b. Click **New Lightning App**.
   c. For **Name**, enter a name such as `Contact Center`.
   d. The **Developer Name** is automatically filled. Click **Next**.
   e. In the App Options settings, select **Console navigation**. Click **Next**.
   f. In the Utility Bar settings page, click **Add Utility Item**.
   g. To add Omni-Channel to the utility bar in the console footer, select **Omni-Channel**.
   h. After you add Omni-Channel to the utility bar, the Omni-Channel utility item properties are displayed. Leave them as is and click **Next**.
   i. Add Voice Calls as a navigation item in the console.
   j. Click **Save & Finish**.

Set Up Service Cloud Voice for Partner Telephony in Your Org

When Service Cloud Voice for Partner Telephony licenses are added to an org, Salesforce admins in the org can open the Partner Telephony Setup page and follow the steps to set up their contact center. This guide describes how you can customize that experience.

❗ Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.



SETUP
Service Cloud Voice

SERVICE CLOUD VOICE

1 Get Ready for Service Cloud Voice
Set up these prerequisites in your org. If you've already got them, skip ahead to step 2.

Create a Custom Domain ✓
Use My Domain to create a custom login URL that you can use to configure single sign-on (SSO) for agents. If you change your domain after creating a contact center, you must restart your Voice implementation.

Enable Omni-Channel ✓
Turn on Omni-Channel so agents can answer and make calls.

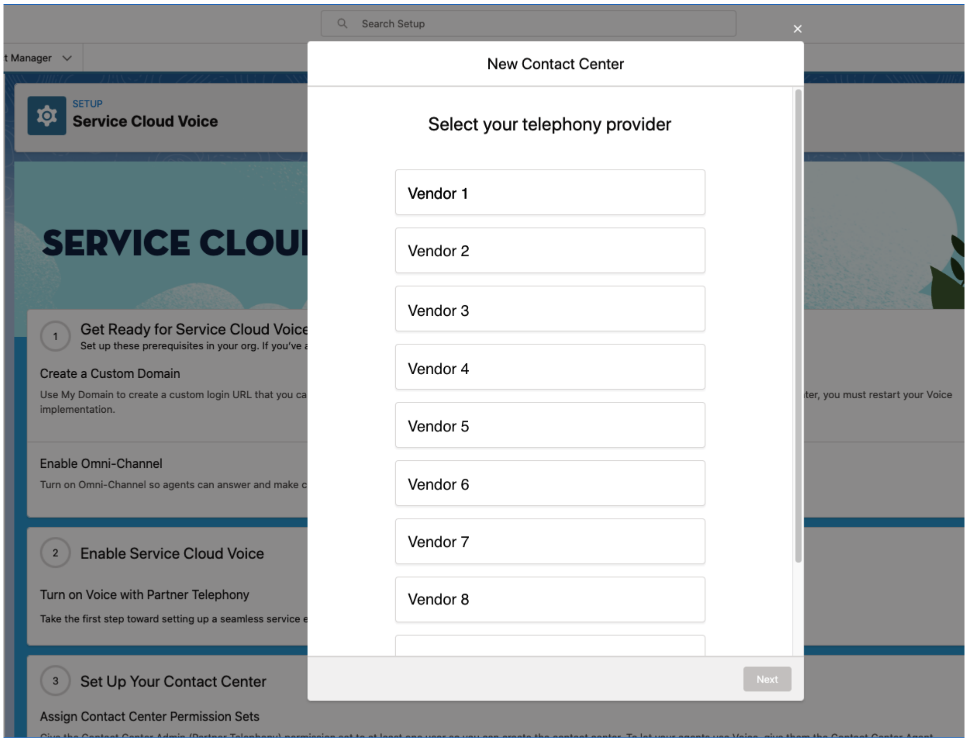
2 Enable Service Cloud Voice

Turn on Voice with Partner Telephony — Enabled
Take the first step toward setting up a seamless service experience with your preferred telephony provider.

Customers can install the managed package from the App Exchange or from an external URL provided by the telephony partner. Telephony partners can skip installing the managed package from the App Exchange, because they would install the managed package in their scratch orgs in Create an SFDX Project on page 11. Telephony partners can also follow these steps to set up contact centers in their scratch orgs.

After the admin completes the initial step of enabling Omni-Channel, they can enable Voice in their org. The next steps require assigning the contact center permissions and creating the contact center.

1. Both customers and telephony partners can follow the setup wizard to create a contact center.



New Contact Center

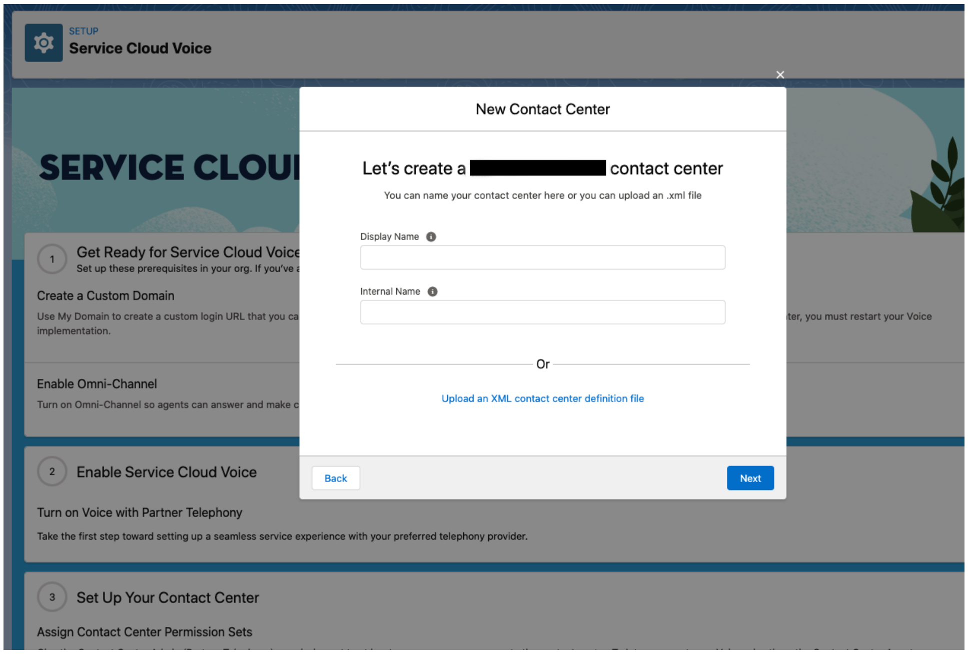
Select your telephony provider

Vendor 1
Vendor 2
Vendor 3
Vendor 4
Vendor 5
Vendor 6
Vendor 7
Vendor 8

Next

Search Setup
SETUP
Service Cloud Voice
SERVICE CLOU
3 Set Up Your Contact Center
Assign Contact Center Permission Sets

2. After selecting the available vendor from the installed packages, there are two options. Either enter the contact center display name and internal name, or import an XML file to create a contact center.



SETUP
Service Cloud Voice
SERVICE CLOU
1 Get Ready for Service Cloud Voic
Create a Custom Domain
Enable Omni-Channel
2 Enable Service Cloud Voice
Turn on Voice with Partner Telephony
3 Set Up Your Contact Center
Assign Contact Center Permission Sets

New Contact Center

Let's create a [redacted] contact center

You can name your contact center here or you can upload an .xml file

Display Name

Internal Name

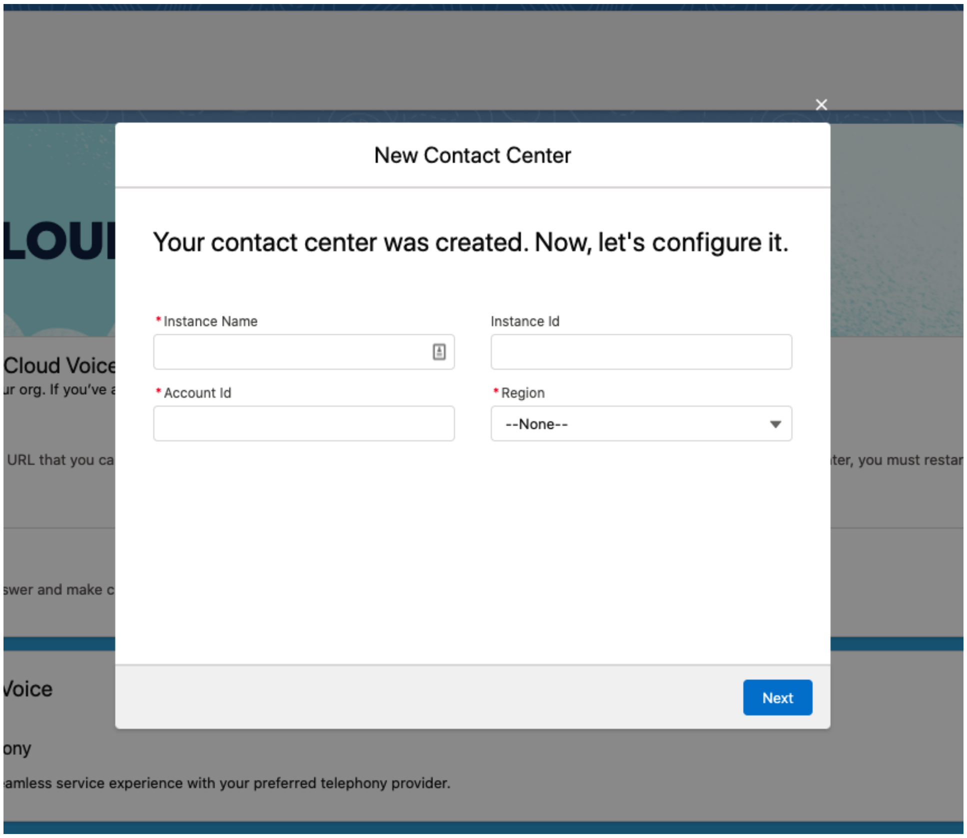
Or

Upload an XML contact center definition file

Note: As you type text into the text field, the **Upload an XML definition file** link becomes disabled. Clear the text field to re-enable the link.

The next several steps describe the flow if you choose to enter the contact center name (rather than importing the XML).

3. The next setup step shows custom settings based on a custom object you can specify in the ConversationVendorInfo setup object. If you don't have a custom object specified in ConversationVendorInfo, this step is skipped.


New Contact Center

Your contact center was created. Now, let's configure it.

* Instance Name:

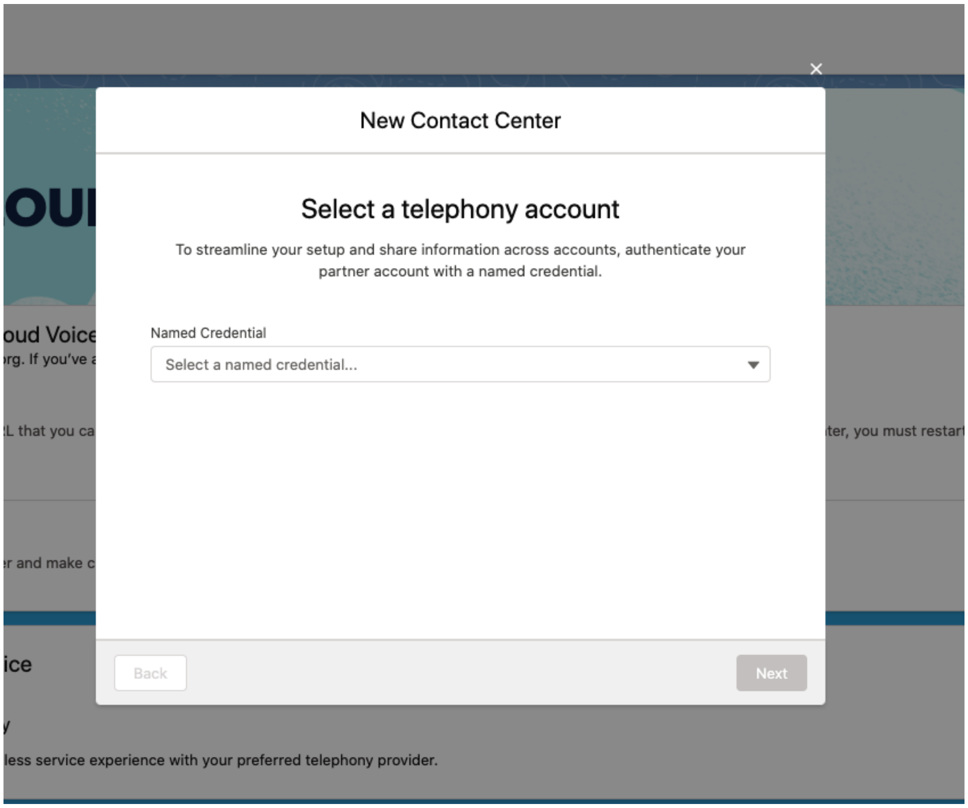
Instance Id:

* Account Id:

* Region: --None--

If the partner wants to allow customers to provide some non-standard settings, the partner can create a custom object and specify it in the ConversationVendorInfo object `customConfig` field. In this step, the setup wizard loads all the fields from the default layout to allow customers to provide values for settings. It creates a record of this custom object and stores the record ID in the contact center. When reps log in to the contact center, the record details are loaded as part of the contact center settings.

4. If the partner wants to have an advanced feature that would require the server-to-server communication with a named credential, it must be declared in the ConversationVendorInfo object `namedCredentialSupported` field. If the partner doesn't support this feature, this step is skipped.



New Contact Center

Select a telephony account

To streamline your setup and share information across accounts, authenticate your partner account with a named credential.

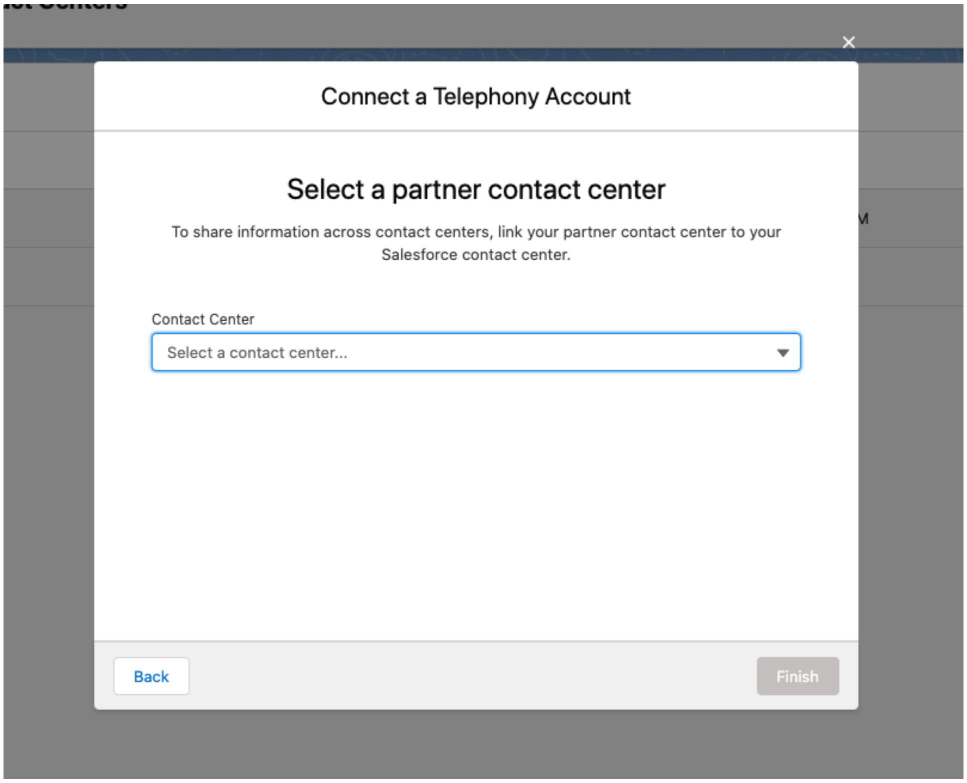
Named Credential

Select a named credential...

This functionality can be used for Einstein Conversation Insights, user syncing, or retrieving the external queue ID for queue mapping. These features require named credentials for the server-to-server authentication. In this step, the screen allows customers to choose which named credentials to use. It shows all the available named credentials in this org, either from local creation or an installed managed package. After clicking the **Next** button, it invokes the connect method from the Apex class that implements the `service_cloud_voice.PartnerConnector` on page 125 interface. If the response is successful, it stores the named credential ID in the contact center.

a. If the partner also supports the rep SSO authentication, it must be declared in the ConversationVendorInfo object `agentSSOSupported` field. You must also provide an Apex class, which implements the `service_cloud_voice.PartnerSSO` on page 125 interface. This step invokes the Apex class methods to get the required parameters and create the connected app to set up the SAML identity provider.

b. If the partner supports named credentials, and there are multiple contact center instances in the vendor's system, and you want to let the customer choose which one to connect to, this information must be declared in the ConversationVendorInfo object `partnerContactCenterListSupported` field. This information also must be declared in the connect method from the Apex class that implements the `service_cloud_voice.PartnerConnector` on page 125 interface. This interface returns the list of the contact centers, with external IDs and labels. This step allows the customer to choose which external contact center to connect to. If this feature isn't supported, this step is skipped.



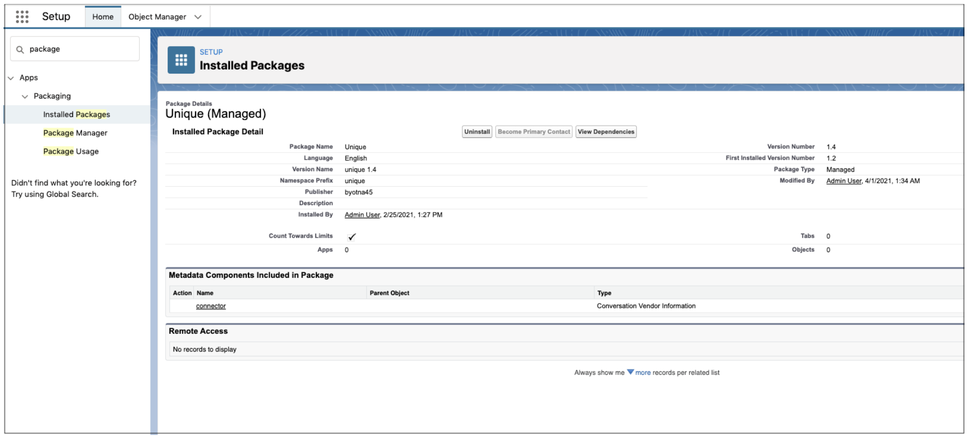
Connect a Telephony Account

Select a partner contact center

To share information across contact centers, link your partner contact center to your Salesforce contact center.

Contact Center: Select a contact center...

5. When importing XML, the content contains two mandatory sections and some optional, vendor-specific sections.
   - General Info Section:
     - Contains the Display Name and Internal Name. Use alphanumeric characters for the internal name of contact center. The internal name must be unique.
     - Set the `reqVendorInfoApiName` to <namespace prefix of managed package>__<developer name of the Conversation Vendor Info component>. You can check the namespace prefix and developer name from View Components inside the managed package.



Installed Packages

Package Details

Unique (Managed)

Installed Package Detail

Package Name	Unique	Version Number	1.4
Language	English	First Installed Version Number	1.2
Version Name	unique 1.4	Package Type	Managed
Namespace Prefix	unique	Modified By	Admin User, 4/1/2021, 1:34 AM
Publisher	byotna45		
Description			
Installed By	Admin User, 2/25/2021, 1:27 PM		
Count Towards Limits	✓	Tabs	0
Apps	0	Objects	0

Metadata Components Included in Package

Action	Name	Parent Object	Type
	connector		Conversation Vendor Information

Remote Access

No records to display

Always show me ▼ more records per related list

   - SCV Settings Section:

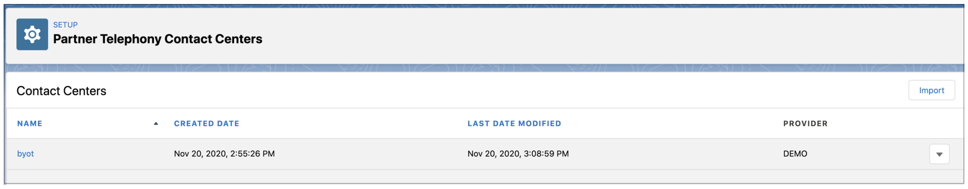
  - Set the `reqTelephonyIntegrationCertificate` to the certificate as generated in Generate a Self-Signed Certificate with OpenSSL on page 16. Customers can use this info as provided by the telephony partner.
  - Update `reqLongDistPrefix` to the desired area code prefix. The Omni-Channel softphone uses this area code for outbound calls.

- Partner Settings Sections:
  - Telephony partners can include their specific configurations in the rest of the sections with a name other than reqGeneralInfo/reqHvcc. Customers don't must change the configurations specified by the telephony partner.

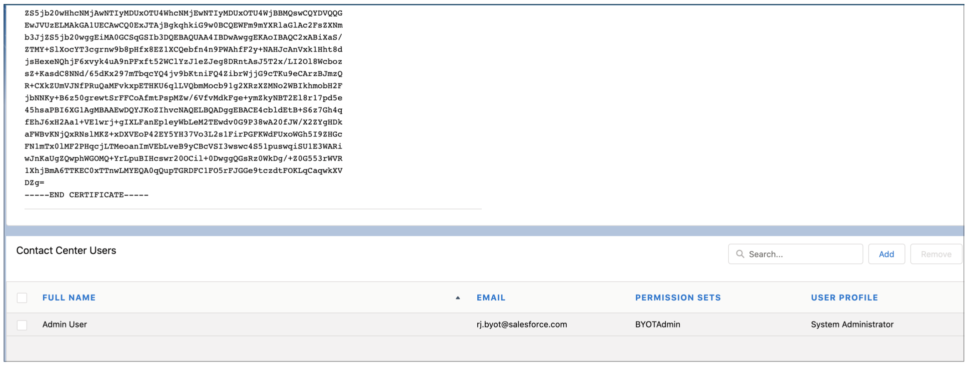
- There's a maximum limit of 500 sections per XML file and 1,000 items per section.

```
<callCenter>

    <!-- General Info Section. This is a required section. -->
    <section sortOrder="0" name="reqGeneralInfo" label="General Information">
        <item sortOrder="1" name="reqDisplayName" label="Display Name">Quick Start
Partner Telephony</item>
        <item sortOrder="0" name="reqInternalName"
label="InternalName">quickStartPartnerTelephony</item>

        <!-- Provide Full API Name of the Conversation Vendor Info Component present
 in installed vendor managed package -->
        <item sortOrder="2" name="reqVendorInfoApiName" label="Conversation Vendor
Info Developer Name">quickStartPT__quickStartPartnerTelephony</item>
    </section>

    <!-- SCV Settings. This is a required section. -->
    <section sortOrder="1" name="reqHvcc" label="SCV Settings">
        <item sortOrder="0"
            name="reqTelephonyIntegrationCertificate" label="Telephony Integration
Certificate">-----BEGIN CERTIFICATE-----
MIIC/jCCAeYCCQCIBPHtpQA4CjANBgkqhkiG9w0BAQsFADBBMQswCQYDVQQGEwJV
UzELMAkGA1UECAwCQ0ExJTAjBgkqhkiG9w0BCQEWFm9mYXRlaGlAc2FsZXNmb3Jj
ZS5jb20wHhcNMjAwNTIyMDUxOTU4WhcNMjEwNTIyMDUxOTU4WjBBMQswCQYDVQQG
EwJVUzELMAkGA1UECAwCQ0ExJTAjBgkqhkiG9w0BCQEWFm9mYXRlaGlAc2FsZXNm
b3JjZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC2xABiXaS/
ZTMY+SlXocYT3cgrnw9b8pHfx8EZ1XCQebfn4n9PWAhfF2y+NAHJcAnVxk1Hht8d
jsHexeNQhjF6xvyk4uA9nPFxft52WClYzJ1eZJeg8DRntAsJ5T2x/LI2Ol8Wcboz
sZ+KasdC8NNd/65dKx297mTbqcYQ4jv9bKtniFQ4ZibrWjjG9cTKu9eCArzBJmzQ
R+CXkZUmVJNfPRuQaMFvkxpETHKU6qlLVQbmMocb91g2XRzXZMNo2WBIkhmobH2F
jbNNKy+B6z50grewtSrFFCoAfmtPspMZw/6VfvMdkFge+ymZkyNBT2El8r17pd5e
45hsaPBI6XG1AgMBAAEwDQYJKoZIhvcNAQELBQADggEBACE4cbldEtB+S6z7Gh4q
fEhJ6xH2Aa1+VE1wrj+gIXLFanEp1eyWbLeM2TEwdv0G9P38wA20fJW/X2ZYgHDk
aFWBvKNjQxRNslMKZ+xDXVEoP42EY5YH37Vo3L2s1FirPGFKWdFUxoWGh5I9ZHGc
FN1mTx0lMF2PHqcjLTMeoanImVEbLveB9yCBcVSI3wswc4S51puswqiSU1E3WARi
wJnKaUgZQwphWGOMQ+YrLpuBIHcswr20OCil+0DwggQGsRz0WkDg/+Z0G553rWVR
1XhjBmA6TTKEC0xTTnwLMYEQA0qQupTGRDFC1FO5rFJGGe9tczdtFOKLqCaqwkXV
DZg=
-----END CERTIFICATE-----</item>
        <item sortOrder="1" name="reqLongDistPrefix" label="Long Distance
Prefix">+1</item>
    </section>
    <!-- All below sections below contain vendor settings-->
    <section sortOrder="2" name="providerSettings1" label="Button Assignment">
        <item sortOrder="0" name="custom2" label="custom2">Custom2 </item>
        <item sortOrder="2" name="custom0" label="custom0">custom0</item>
        <item sortOrder="1" name="custom1" label="custom1">Custom1</item>

    </section>
    <section sortOrder="3" name="providerSettings2" label="Dialing Options">
        <item sortOrder="2" name="custom0" label="custom0">custom0</item>
        <item sortOrder="0" name="custom2" label="custom2">Custom2</item>
        <item sortOrder="1" name="custom1" label="custom1">Custom1</item>

    </section>
</callCenter>
```

6. After the setup flow is completed, the contact center appears in the Partner Telephony Contact Centers node and the admin can add users to the contact center.



NAME	CREATED DATE	LAST DATE MODIFIED	PROVIDER
byot	Nov 20, 2020, 2:55:26 PM	Nov 20, 2020, 3:08:59 PM	DEMO

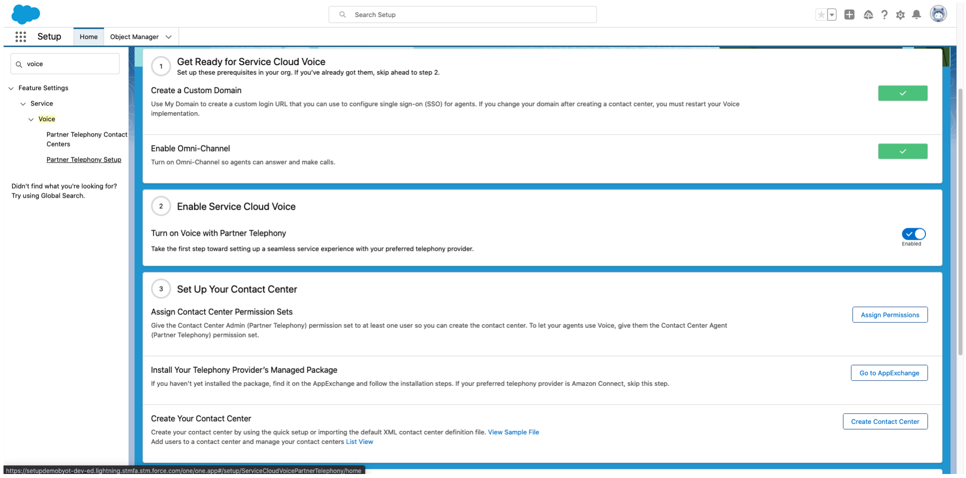


```
ZS5jb20wHhcNMjAwNTIyMDUxOTU4WhcNMjEwNTIyMDUxOTU4WjBBMQswCQYDVQQG
EwJVUzELMAkGA1UECAwCQ0ExJTAjBgkqhkiG9w0BCQEWFm9mYXRlaGlAc2FsZXNm
b3JjZS5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC2xABiXaS/
ZTMY+SlXocYT3cgrnw9b8pHfx8EZ1XCQebfn4n9PWAhfF2y+NAHJcAnVxk1Hht8d
jsHexeNQhjF6xvyk4uA9nPFxft52WClYzJ1eZJeg8DRntAsJ5T2x/LI2Ol8Wcboz
sZ+KasdC8NNd/65dKx297mTbqcYQ4jv9bKtniFQ4ZibrWjjG9cTKu9eCArzBJmzQ
R+CXkZUmVJNfPRuQaMFvkxpETHKU6qlLVQbmMocb91g2XRzXZMNo2WBIkhmobH2F
jbNNKy+B6z50grewtSrFFCoAfmtPspMZw/6VfvMdkFge+ymZkyNBT2El8r17pd5e
45hsaPBI6XG1AgMBAAEwDQYJKoZIhvcNAQELBQADggEBACE4cbldEtB+S6z7Gh4q
fEhJ6xH2Aa1+VE1wrj+gIXLFanEp1eyWbLeM2TEwdv0G9P38wA20fJW/X2ZYgHDk
aFWBvKNjQxRNslMKZ+xDXVEoP42EY5YH37Vo3L2s1FirPGFKWdFUxoWGh5I9ZHGc
FN1mTx0lMF2PHqcjLTMeoanImVEbLveB9yCBcVSI3wswc4S51puswqiSU1E3WARi
wJnKaUgZQwphWGOMQ+YrLpuBIHcswr20OCil+0DwggQGsRz0WkDg/+Z0G553rWVR
1XhjBmA6TTKEC0xTTnwLMYEQA0qQupTGRDFC1FO5rFJGGe9tczdtFOKLqCaqwkXV
DZg=
-----END CERTIFICATE-----
```

□	FULL NAME	EMAIL	PERMISSION SETS	USER PROFILE
□	Admin User	rj.byot@salesforce.com	BYOTAdmin	System Administrator

Other details about the setup process:

- The admin clicks **Create Contact Center** to launch the wizard.


Search Setup
Setup
Home
Object Manager
voice
Feature Settings
Service
Voice
Partner Telephony Contact Centers
Partner Telephony Setup
Didn't find what you're looking for? Try using Global Search.

1 Get Ready for Service Cloud Voice

Set up these prerequisites in your org. If you've already got them, skip ahead to step 2.

Create a Custom Domain ✓

Use My Domain to create a custom login URL that you can use to configure single sign-on (SSO) for agents. If you change your domain after creating a contact center, you must restart your Voice implementation.

Enable Omni-Channel ✓

Turn on Omni-Channel so agents can answer and make calls.

2 Enable Service Cloud Voice

Turn on Voice with Partner Telephony — Enabled

Take the first step toward setting up a seamless service experience with your preferred telephony provider.

3 Set Up Your Contact Center

Assign Contact Center Permission Sets — Assign Permissions

Give the Contact Center Admin (Partner Telephony) permission set to at least one user so you can create the contact center. To let your agents use Voice, give them the Contact Center Agent (Partner Telephony) permission set.

Install Your Telephony Provider's Managed Package — Go to AppExchange

If you haven't yet installed the package, find it on the AppExchange and follow the installation steps. If your preferred telephony provider is Amazon Connect, skip this step.

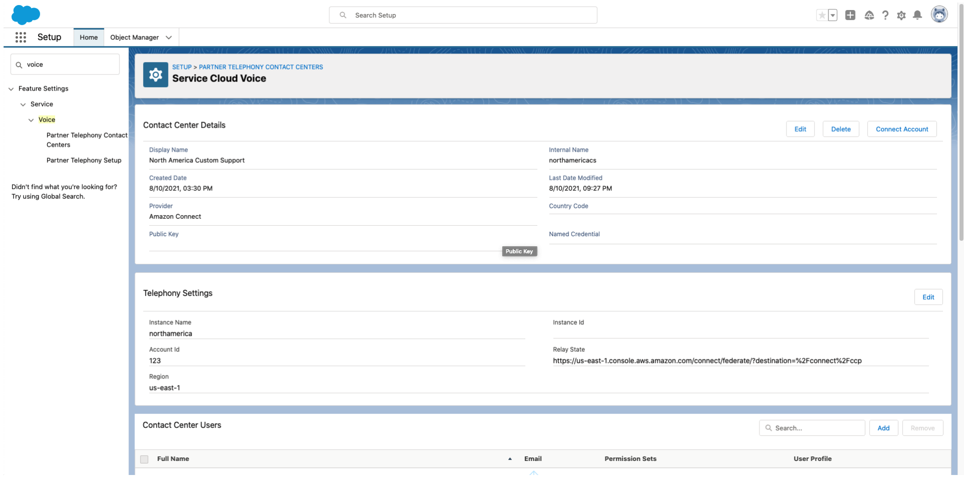
Create Your Contact Center — Create Contact Center

Create your contact center by using the quick setup or importing the default XML contact center definition file. View Sample File

Add users to a contact center and manage your contact centers List View

https://setupdemobyot-dev-ed.lightning.stmfa.stm.force.com/one/one.app#/setup/ServiceCloudVoicePartnerTelephony/home

- You can change the values on the contact center details page after the contact center is created.
- If the setup flow is canceled before completion, the steps can still be completed on the contact center details page. The custom settings can be edited in the second section of the contact center details page. The connect button is on the top-right corner, which allows the admin to launch the wizard to select the named credential to connect to the telephony account. In the create contact center wizard, it doesn't ask the public key, so the admin must update the contact center to provide the public key.



Service Cloud Voice

Contact Center Details

Display Name: North America Custom Support

Internal Name: northamericacs

Created Date: 8/10/2021, 03:30 PM

Last Date Modified: 8/10/2021, 09:27 PM

Provider: Amazon Connect

Country Code:

Public Key:

Named Credential:

Telephony Settings

Instance Name: northamerica

Instance Id:

Account Id: 123

Relay State: https://us-east-1.console.aws.amazon.com/connect/federate/?destination=%2Fconnect%2Fccp

Region: us-east-1

Contact Center Users

□	Full Name ▲	Email	Permission Sets	User Profile

- After this step is completed, the contact center appears in the Partner Telephony Contact Centers node and the admin can add users to the contact center.

CHAPTER 5 Add Support for Voice Resiliency

Voice resiliency ensures that calls can still go through when the number of conversations is over limit or when the background service is affected.

The voice resiliency service allows customers to continue using basic voice features such as inbound calling, outbound calling, call transfers, and callbacks when the conversation entities creation fails. During this period, certain features and functionalities may be impacted. See the Voice Resiliency for Service Cloud Voice knowledge article for more information about voice resiliency.

If your environment isn't set up correctly for voice resiliency, you may experience missing calls. To support voice resiliency, make sure that in the event of a call failure the call still gets routed to the rep or queue.

Configure the following steps to ensure your environment is set up correctly for voice resiliency.

1. If the Create a Voice Call Record API (`/telephony/v1/voiceCalls`) fails, perform up to three retries. Retries may fail if, for example, you've reached the conversation limit.
2. If the retries continue to fail, still route the call to the rep or queue, and send the CALL_STARTED event through the connector.

CHAPTER 6 Connect Your Telephony System to Salesforce

In this chapter ...

The demo connector is a quick and effective way to get your telephony system communicating with your Salesforce org. This demo connector uses the Connector API, which is the interface between your telephony system and Salesforce.

❗ Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that’s not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference: Get Started with the Service Cloud Connector API

Use the Demo Connector

The demo connector is a sample application for partner telephony systems that integrate with Service Cloud Voice. It demonstrates an optimal Voice implementation based on a group of telephony API mocks. It also includes a voice call simulation tool that you can use to test call actions such as making and answering calls and using phone controls.

Start the demo connector

The `byo-demo-connector` is provided as part of the Demo Connector in GitHub.

1. Clone the git repo, and install the NPM dependencies.

```
$ git clone git@github.com:salesforce-misc/byo-demo-connector.git
$ cd byo-demo-connector
$ npm install
```

2. Add a private key generated from Generate a Self-Signed Certificate with OpenSSL on page 16.

```
{byo-demo-connector}
                |__src
                    |__server
                            |__private.key
```

3. Launch the tool.

```
$ npm start
```

By default, the web server runs in SSL on port 8080. The **adapterUrl** in your contact center points to this web server (for example, `https://www.myTelephonyDemo.com:8080`).

The previous command also starts the SCV REST APIs connector on default port 3030. It's used for creating voice calls, transcription, and recording.

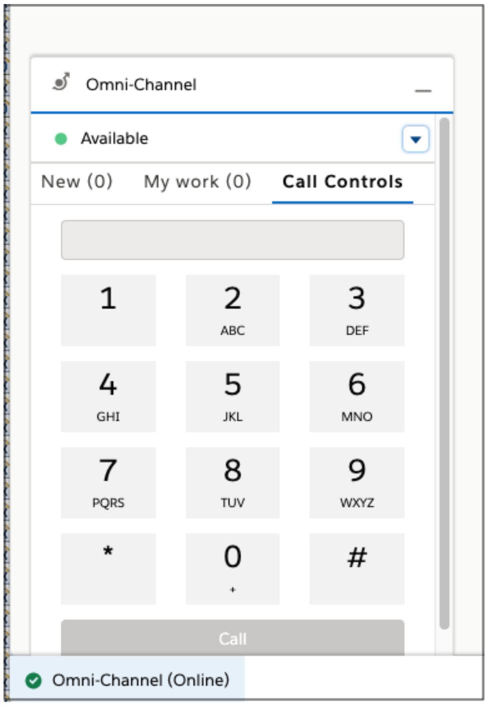
You can run them on separate terminals using the following commands.

```
$ npm run client
$ npm run server
```

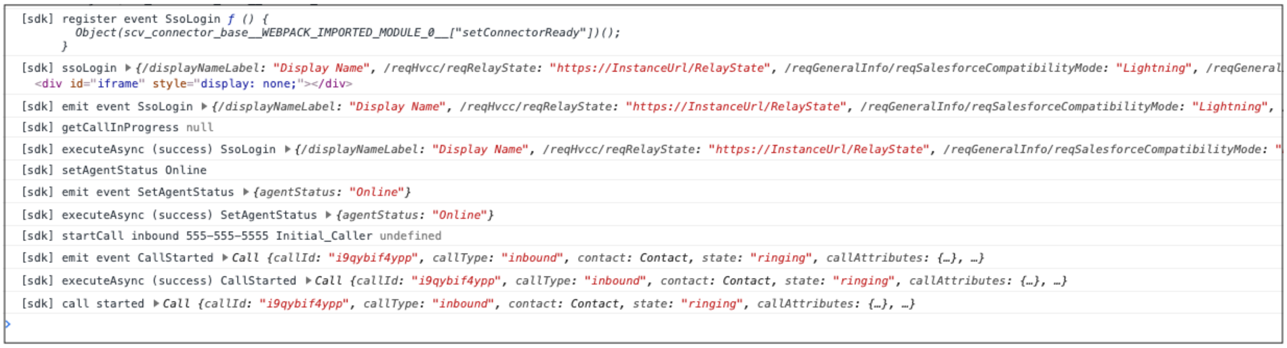
Verify That the Connector is Loaded

Test that the connector is working.

1. The `byo-demo-connector` app uses a self-signed certificate, so you must get your web browser to accept a self-signed certificate.
   a. Open the `byo-demo-connector` app URL (for example, `https://serverURL:8080/remote.html`) in a separate tab.
   b. Click through any warnings for untrusted certificates.
2. Log in to Salesforce as one of the users that were added to the contact center.
3. Log in to the Omni-Channel utility and change the rep status to **Available**.



Open the Browser Debugger and make sure that you see messages from the connector. The messages start with `[sdk]` or `[connector]`.



```
[sdk] register event SsoLogin ƒ () {
        Object(scv_connector_base__WEBPACK_IMPORTED_MODULE_0__["setConnectorReady"])();
      }
[sdk] ssoLogin ▶{/displayNameLabel: "Display Name", /reqHvcc/reqRelayState: "https://InstanceUrl/RelayState", /reqGeneralInfo/reqSalesforceCompatibilityMode: "Lightning", /reqGeneral
  <div id="iframe" style="display: none;"></div>
[sdk] emit event SsoLogin ▶{/displayNameLabel: "Display Name", /reqHvcc/reqRelayState: "https://InstanceUrl/RelayState", /reqGeneralInfo/reqSalesforceCompatibilityMode: "Lightning",
[sdk] getCallInProgress null
[sdk] executeAsync (success) SsoLogin ▶{/displayNameLabel: "Display Name", /reqHvcc/reqRelayState: "https://InstanceUrl/RelayState", /reqGeneralInfo/reqSalesforceCompatibilityMode: "
[sdk] setAgentStatus Online
[sdk] emit event SetAgentStatus ▶{agentStatus: "Online"}
[sdk] executeAsync (success) SetAgentStatus ▶{agentStatus: "Online"}
[sdk] startCall inbound 555-555-5555 Initial_Caller undefined
[sdk] emit event CallStarted ▶Call {callId: "i9qybif4ypp", callType: "inbound", contact: Contact, state: "ringing", callAttributes: {…}, …}
[sdk] executeAsync (success) CallStarted ▶Call {callId: "i9qybif4ypp", callType: "inbound", contact: Contact, state: "ringing", callAttributes: {…}, …}
[sdk] call started ▶Call {callId: "i9qybif4ypp", callType: "inbound", contact: Contact, state: "ringing", callAttributes: {…}, …}
>
```

SEE ALSO:

GitHub: Demo Connector

Use the Connector API

The Connector API is the interface between your partner messaging or telephony system and your Salesforce org. This API allows you to pass information to Salesforce, and to receive events back from Salesforce.

Important: The Service Cloud Connector API is for partners who are implementing Bring Your Own Channel for Messaging, Bring Your Own Channel for CCaaS, or Service Cloud Voice with Partner Telephony.

For Service Cloud Voice with Partner Telephony, the demo connector on page 32 is a working example that uses the Connector API. For details on using the Connector API, you can review our reference documentation.

The Service Cloud Connector API contains two parts:

1. Base Connector API
2. Connector API Methods

SEE ALSO:

Service Cloud Connector API Reference

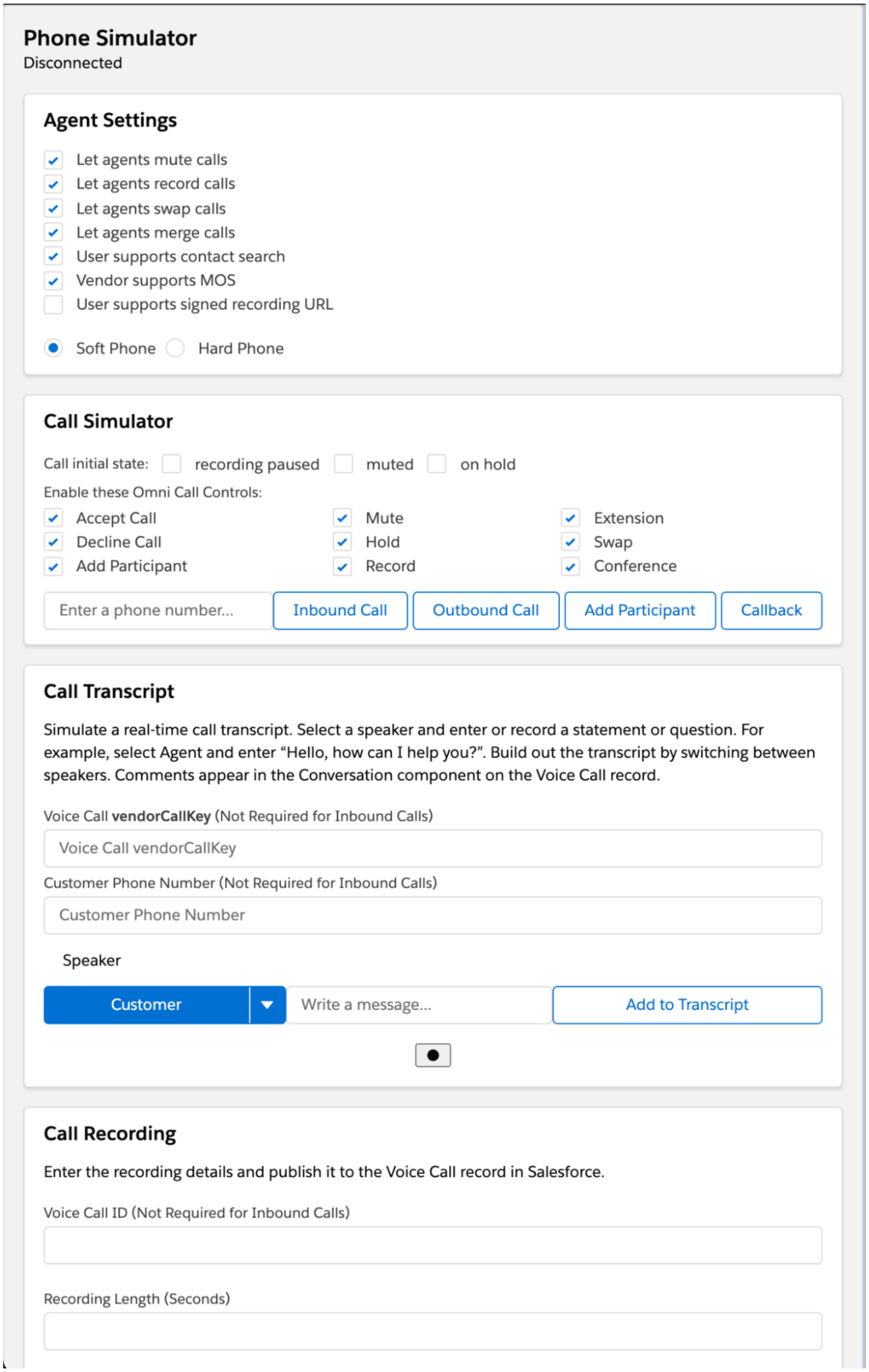
Service Cloud Connector API Reference: Get Started with the Service Cloud Connector API

Test Your Implementation with the Voice Call Simulator

The Voice Call Simulator component helps you get comfortable with Service Cloud Voice. As you set up Service Cloud Voice, use the simulator to walk through a variety of call scenarios.

To get started, load the demo connector project as described in Use the Demo Connector on page 32. Then, open the Voice Call Simulator by navigating to `https://:8080/remote.html`. We recommend leaving the simulator open in one browser tab and the service console in another tab.

The following screens appear with the simulator.



Phone Simulator

Disconnected

Agent Settings

☑ Let agents mute calls
☑ Let agents record calls
☑ Let agents swap calls
☑ Let agents merge calls
☑ User supports contact search
☑ Vendor supports MOS
□ User supports signed recording URL

● Soft Phone ○ Hard Phone

Call Simulator

Call initial state: □ recording paused □ muted □ on hold

Enable these Omni Call Controls:

☑ Accept Call ☑ Mute ☑ Extension
☑ Decline Call ☑ Hold ☑ Swap
☑ Add Participant ☑ Record ☑ Conference

Call Transcript

Simulate a real-time call transcript. Select a speaker and enter or record a statement or question. For example, select Agent and enter "Hello, how can I help you?". Build out the transcript by switching between speakers. Comments appear in the Conversation component on the Voice Call record.

Voice Call **vendorCallKey** (Not Required for Inbound Calls)

Customer Phone Number (Not Required for Inbound Calls)

Speaker

Call Recording

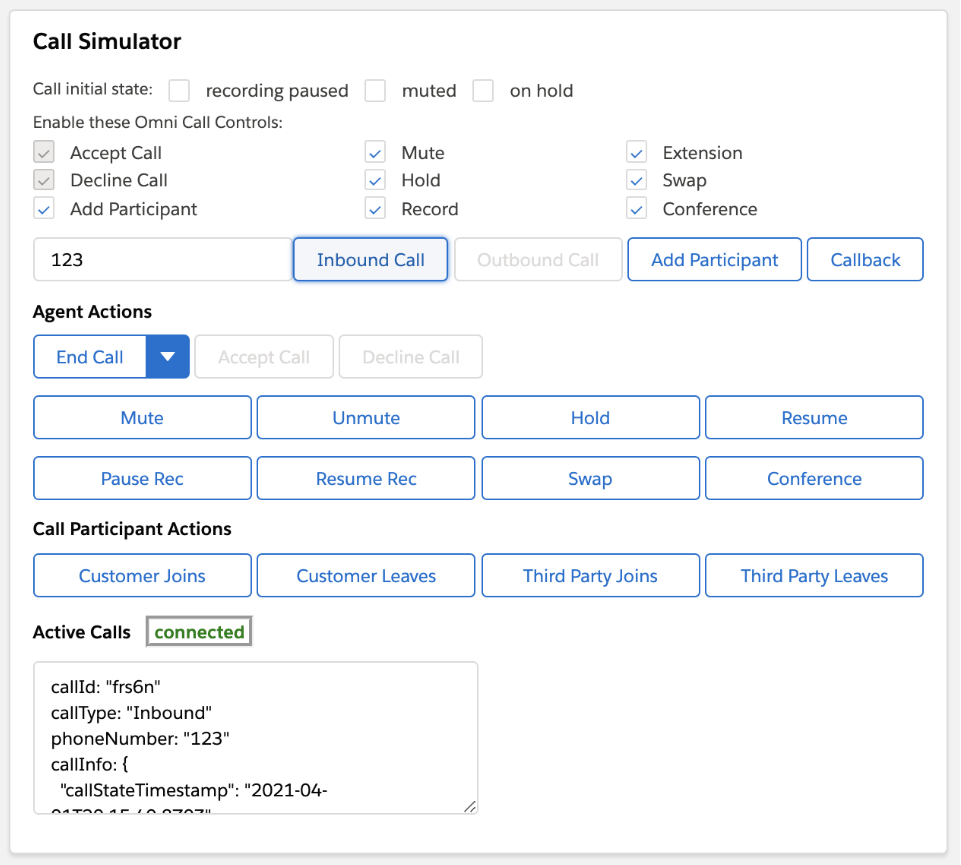
Enter the recording details and publish it to the Voice Call record in Salesforce.

Voice Call ID (Not Required for Inbound Calls)

Recording Length (Seconds)

- **Agent Settings:** Let agents mute, record, swap, merge calls, allow signed recording URLs, support contact search, and support MOS for vendors.

- **Call Transcript:** Simulate a real-time call transcript. Select a speaker and enter or record a statement or question. For example, select Agent and enter, “Hello, how can I help you?”. Build out the transcript by switching between speakers. Comments appear in the Conversation component on the Voice Call record.
- **Call Simulator:** The call simulator serves to simulate both softphone and hardphone devices. With the call simulator, you can do the following.
  - **Call Initial State**: Set the initial state of the call for recording, mute, and hold controls.
  - **Enable Omni Call Controls**: Enable or disable Omni-Channel controls when a call is ongoing.
  - **Inbound Call**: Initiate an inbound call from the number you entered.
  - **Outbound Call**: With Hardphone selected, you can trigger an outbound call.
  - **Add Participant**: With Hardphone selected, you can add a participant.
  - **Callback**: Trigger a callback.
  - After the call is connected, the **Agent Actions** section appears. The agent actions can be used to mute, unmute, hold, resume, pause recording, resume recording, swap, and conference. For an outbound call, Call Participant Actions can be used to add the customer to the call, simulate the customer leaving the call, or simulate a third party joining and leaving the call.
  - **Active Calls**: This section shows the state of the active calls including ringing, connected states. It also shows the details of the active calls:



Call Simulator

Call initial state: □ recording paused □ muted □ on hold

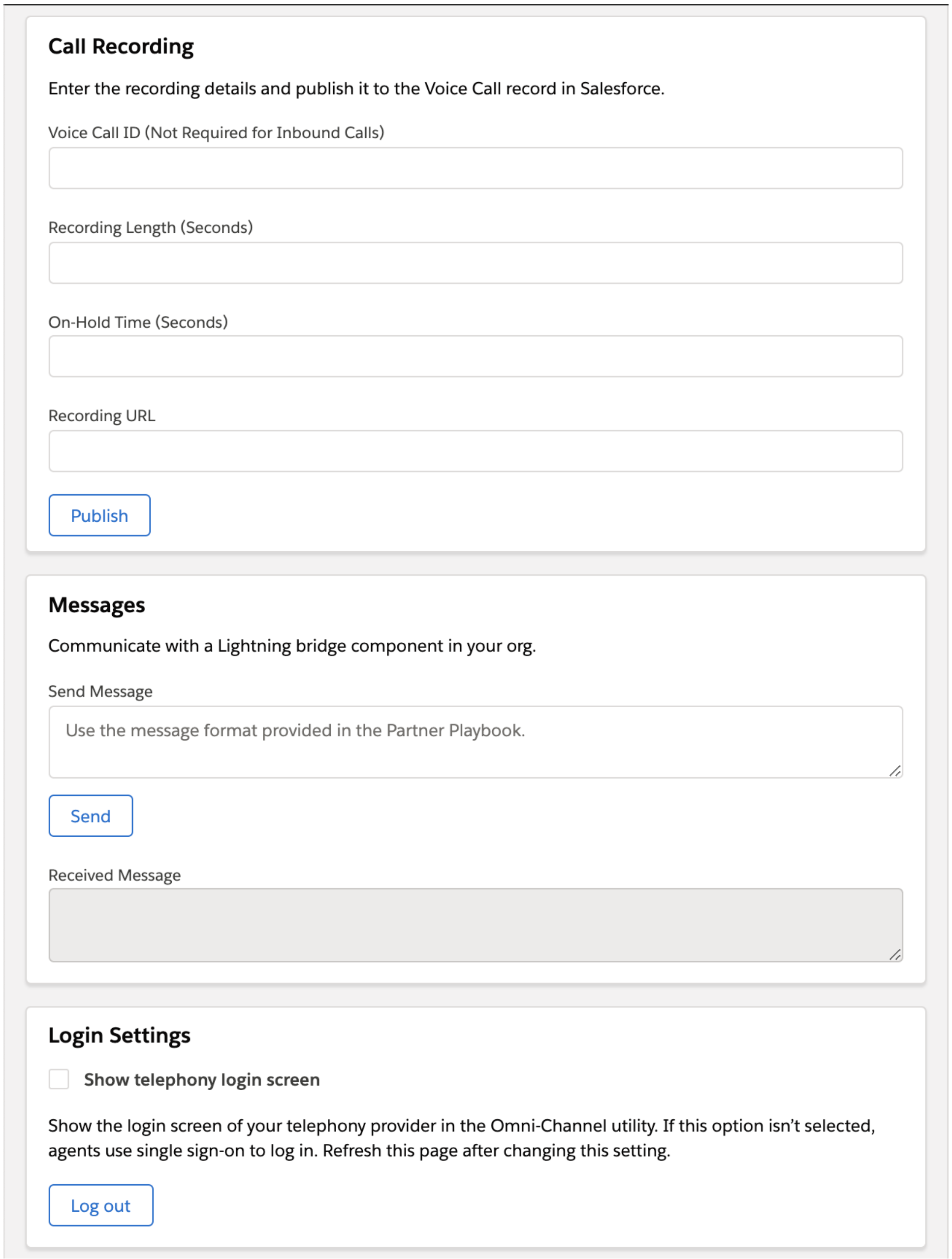
Enable these Omni Call Controls:

☑ Accept Call ☑ Mute ☑ Extension
☑ Decline Call ☑ Hold ☑ Swap
☑ Add Participant ☑ Record ☑ Conference

123

Active Calls connected

```
callId: "frs6n"
callType: "Inbound"
phoneNumber: "123"
callInfo: {
  "callStateTimestamp": "2021-04-
```



Call Recording

Enter the recording details and publish it to the Voice Call record in Salesforce.

Voice Call ID (Not Required for Inbound Calls)

Recording Length (Seconds)

On-Hold Time (Seconds)

Recording URL

Messages

Communicate with a Lightning bridge component in your org.

Send Message

Use the message format provided in the Partner Playbook.

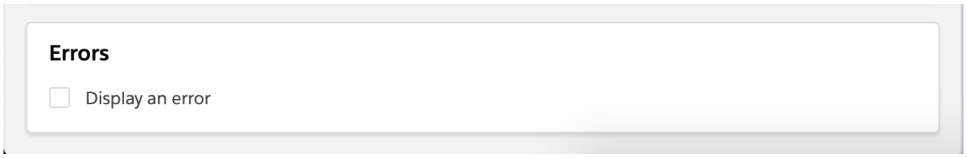
Received Message

Login Settings

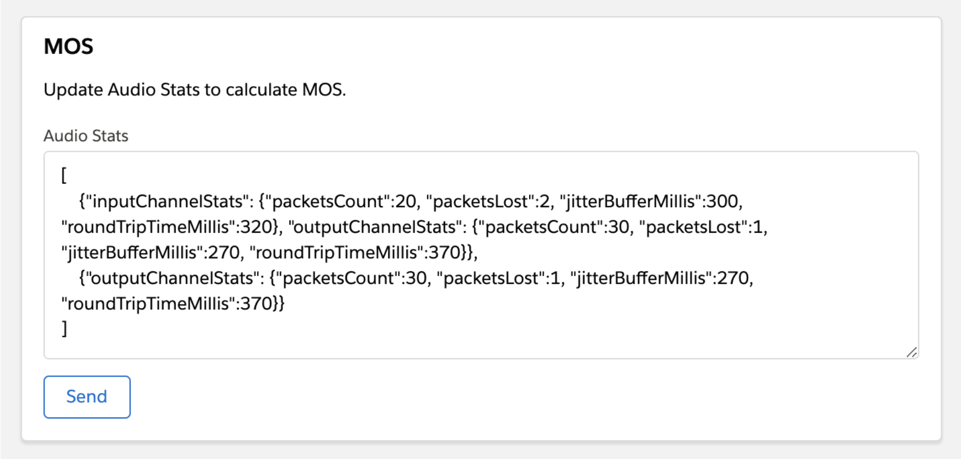
□ Show telephony login screen

Show the login screen of your telephony provider in the Omni-Channel utility. If this option isn't selected, agents use single sign-on to log in. Refresh this page after changing this setting.

- **Login Settings:** Show or hide the telephony login screen in the Omni-Channel utility. Log out from the telephony system. When the telephony login screen is hidden, the rep uses single sign-on to log in.
- **Call Recording:** After you record a call, enter the recording details and publish it to the Voice Call record in Salesforce. The recording URL should point to an audio stream or audio file of the call recording, can't be more than 1,000 characters, and must be fully qualified, meaning that the source is included Trusted URLs with the media-src CSP directive.
- **Messages:** Communicate with a Lightning bridge component in your org. The Received Message field shows the last message received from the component. To learn more, see Use the Lightning Message Service Bridge on page 52.



- **Errors:** Test your error user experience by generating an error in response to any API call. For example, select **Display an error** and mute the call to test the muteCall API error scenario. When you're done, deselect **Display an error**.



```
[
   {"inputChannelStats": {"packetsCount":20, "packetsLost":2, "jitterBufferMillis":300,
"roundTripTimeMillis":320}, "outputChannelStats": {"packetsCount":30, "packetsLost":1,
"jitterBufferMillis":270, "roundTripTimeMillis":370}},
   {"outputChannelStats": {"packetsCount":30, "packetsLost":1, "jitterBufferMillis":270,
"roundTripTimeMillis":370}}
]
```

- **MOS**: Test the Mean Opinion Score (MOS) feature by submitting a JSON object for the audio status.

CHAPTER 7 Set Up Authentication

In this chapter ...

Set up authentication between the Salesforce app and the telephony provider.

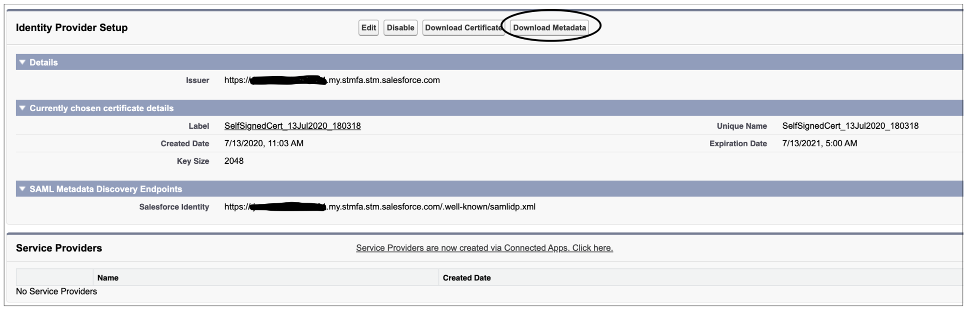
Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Set Up Single Sign-On

Set up single sign-on (SSO) for your solution.

Set Up the SSO Connected App and Salesforce with Identity Provider (IdP)

1. From Setup, enter `Identity Provider` in the Quick Find box, then select **Identity Provider** and enable it. Download the certificate and metadata file.
2. Enter `App Manager` in the Quick Find box. Click **App Manager** > **New connected App**.
3. Create a new connected app and provide the details in Basic information and Web App settings.
   a. Name the connected app.
   b. Enable SAML.
   c. Enter the ACS URL and Entity ID.
   d. Select Subject Type as required and update the name ID format to match the SSO expected.
   e. Select your IdP Certificate from the dropdown options.
   f. Click **Save**.
   g. Click **Manage profile** and select all the profiles for which you want to grant access to use SSO.
4. After enabling Identify Provider, download the metadata file for using Salesforce as IdP. This metadata XML file should be used for setting up SSO.



Identity Provider Setup

▼ Details

Issuer: https://[redacted].my.stmfa.stm.salesforce.com

▼ Currently chosen certificate details

Label: SelfSignedCert_13Jul2020_180318
Unique Name: SelfSignedCert_13Jul2020_180318
Created Date: 7/13/2020, 11:03 AM
Expiration Date: 7/13/2021, 5:00 AM
Key Size: 2048

▼ SAML Metadata Discovery Endpoints

Salesforce Identity: https://[redacted].my.stmfa.stm.salesforce.com/.well-known/samlidp.xml

Service Providers

Service Providers are now created via Connected Apps. Click here.

	Name	Created Date
No Service Providers		

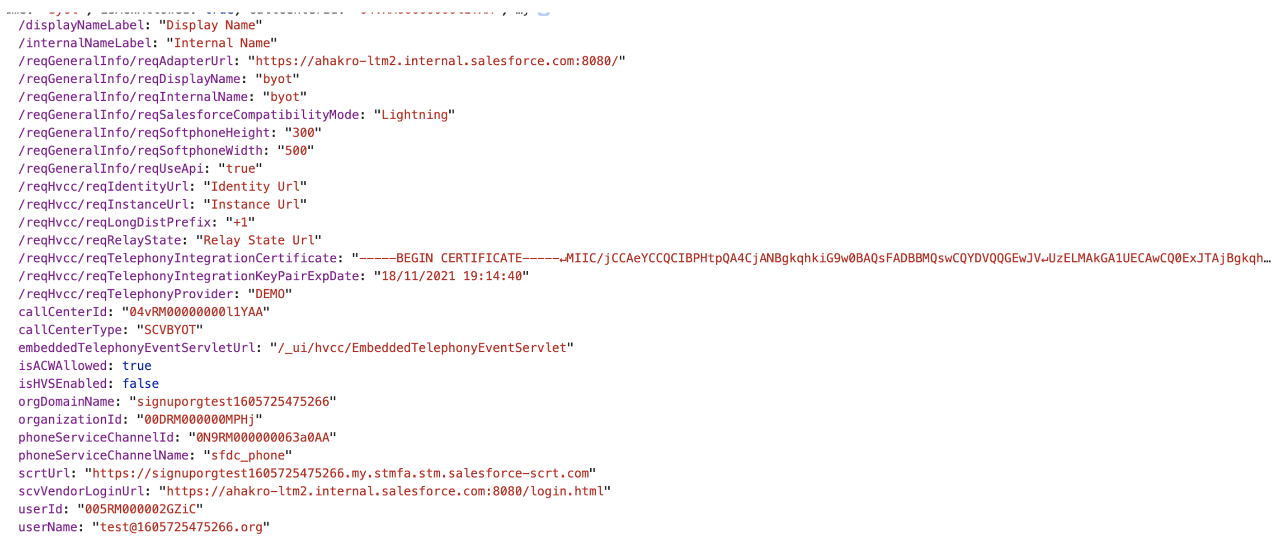
Perform Headless SSO

If a telephony system login page isn't needed, you can use a headless single sign-on (SSO) to your service. Salesforce sends a message to the connector iFrame with the entire contact center configuration as defined in {contactCenterName}.callCenter. Use these details to allow SSO, and notify Salesforce by returning a fulfilled Promise with a value of type `InitResult` when the Promise is successful or rejected.

To set up single sign-on (SSO), configure your Salesforce org as the SAML identity provider. For help, see:

- Set Up SSO Connected App and Salesforce with Identity Provider on page 40
- Salesforce as an Identity Provider

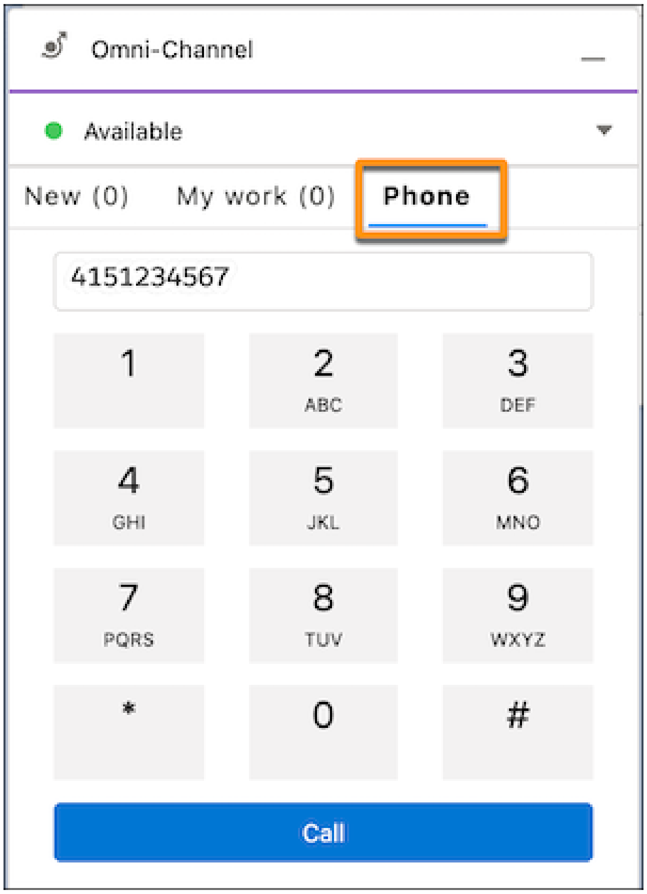
Here's a sample contact center configuration sent from Salesforce during connector init:



```
/displayNameLabel: "Display Name"
/internalNameLabel: "Internal Name"
/reqGeneralInfo/reqAdapterUrl: "https://ahakro-ltm2.internal.salesforce.com:8080/"
/reqGeneralInfo/reqDisplayName: "byot"
/reqGeneralInfo/reqInternalName: "byot"
/reqGeneralInfo/reqSalesforceCompatibilityMode: "Lightning"
/reqGeneralInfo/reqSoftphoneHeight: "300"
/reqGeneralInfo/reqSoftphoneWidth: "500"
/reqGeneralInfo/reqUseApi: "true"
/reqHvcc/reqIdentityUrl: "Identity Url"
/reqHvcc/reqInstanceUrl: "Instance Url"
/reqHvcc/reqLongDistPrefix: "+1"
/reqHvcc/reqRelayState: "Relay State Url"
/reqHvcc/reqTelephonyIntegrationCertificate: "-----BEGIN CERTIFICATE-----↵MIIC/jCCAeYCCQCIBPHtpQA4CjANBgkqhkiG9w0BAQsFADBBMQswCQYDVQQGEwJV↵UzELMAkGA1UECAwCQ0ExJTAjBgkqh…
/reqHvcc/reqTelephonyIntegrationKeyPairExpDate: "18/11/2021 19:14:40"
/reqHvcc/reqTelephonyProvider: "DEMO"
callCenterId: "04vRM00000000l1YAA"
callCenterType: "SCVBYOT"
embeddedTelephonyEventServletUrl: "/_ui/hvcc/EmbeddedTelephonyEventServlet"
isACWAllowed: true
isHVSEnabled: false
orgDomainName: "signuporgtest1605725475266"
organizationId: "00DRM000000MPHj"
phoneServiceChannelId: "0N9RM000000063a0AA"
phoneServiceChannelName: "sfdc_phone"
scrtUrl: "https://signuporgtest1605725475266.my.stmfa.stm.salesforce-scrt.com"
scvVendorLoginUrl: "https://ahakro-ltm2.internal.salesforce.com:8080/login.html"
userId: "005RM000002GZiC"
userName: "test@1605725475266.org"
```

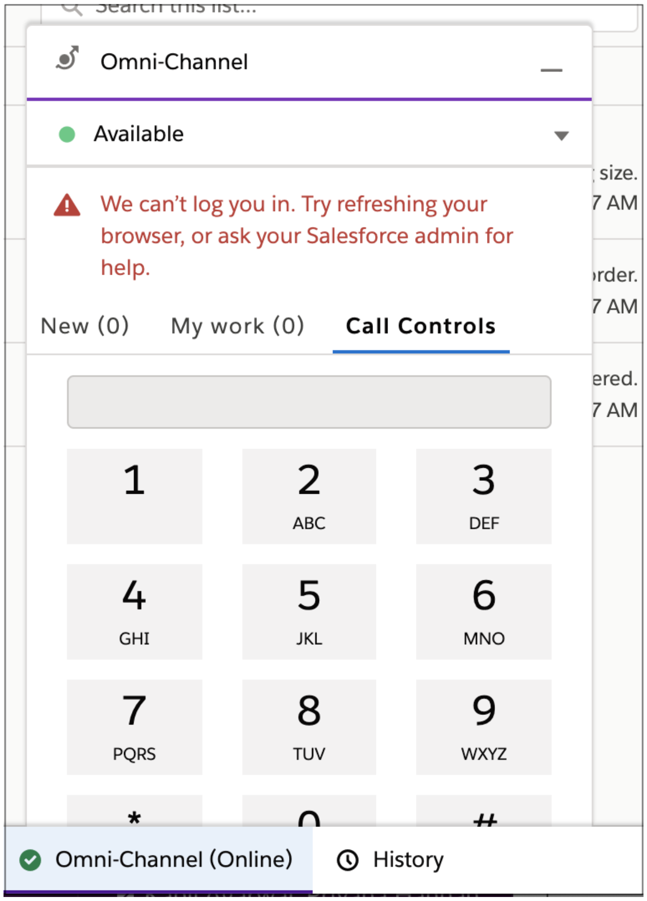
```
init(ssoConfig) {
  const ssoResult = pbx.performSSO(ssoConfig)
  if (ssoResult.success) {
    return Promise.resolve(new InitResult({}));
  } else {
    return Promise.reject("Failed to login");
  }
}
```

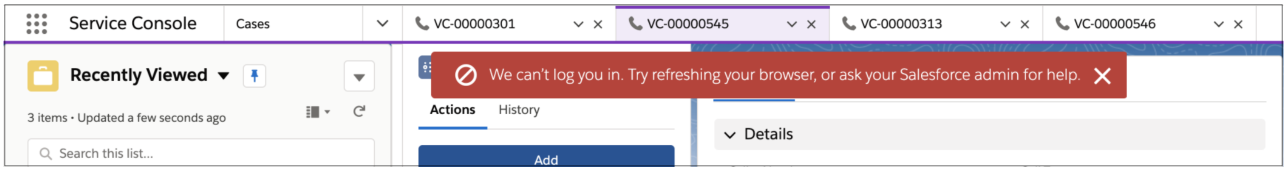
If SSO is successful, the SSO dialer is enabled to allow the rep to make outbound calls.



4151234567

If SSO fails, the base connector dispatches a CAN_NOT_LOG_IN error. The rep receives an error message in the Omni-Channel utility or the Salesforce window, and their keypad remains disabled.


We can’t log you in. Try refreshing your browser, or ask your Salesforce admin for help.


We can’t log you in. Try refreshing your browser, or ask your Salesforce admin for help.

SEE ALSO:

Service Cloud Connector API Reference

Develop a Telephony System Login Page

If needed, create a telephony system login page that appears in the Omni-Channel utility in the Lightning service console.

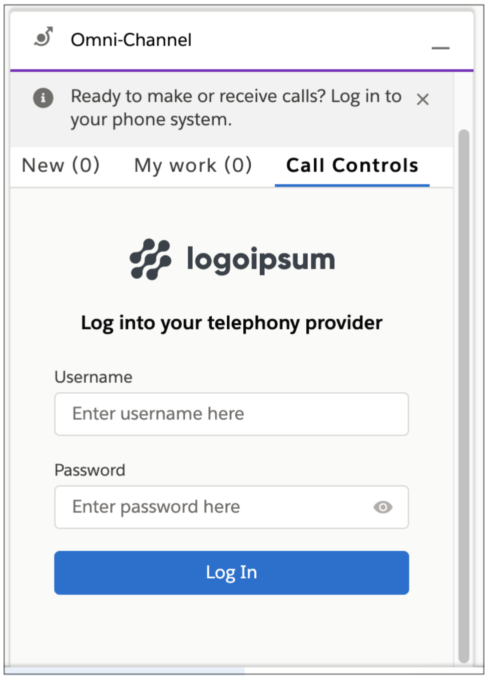
Complete this step if your telephony provider doesn’t support SSO, meaning that reps must log into a telephony system to make and receive calls. The telephony system must maintain the duration and validity of the login session.

1. In your callCenter file, include a `scvVendorLoginUrl` in the `customSettings` field {contactCenterName}.callCenter.

```
<?xml version="1.0" encoding="UTF-8"?>
<CallCenter xmlns="http://soap.sforce.com/2006/04/metadata">
```

```
...

[illegible]
...
</CallCenter>
```

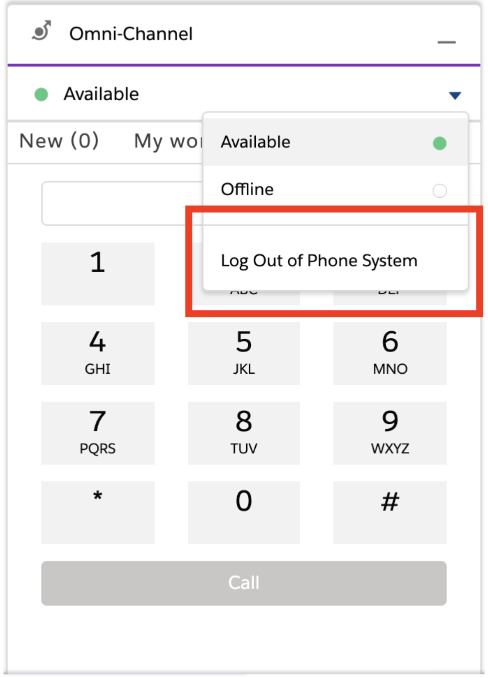
2. In the response to init(), set **showLogin** to true and include an optional number for **loginFrameHeight** in the InitResult object.
3. Upon initialization, the Omni-Channel utility shows an iframe that is populated with the URL specified in the **scvVendorLoginUrl** field of customSettings in the callCenter file:



4. Publish a Connector API event with the login result to show either a login failure or the dialer.

```
publishEvent({
  eventType: Constants.SHARED_EVENT_TYPE.LOGIN_RESULT,
  payload: new GenericResult({ success: true })
});
```

5. The rep can now log out from Omni-Channel.



6. In the event of a session timeout, the telephony vendor can trigger a logout from the Connector API and use the Connector API publishEvent method. The vendor can provide a loginFrameHeight for the login iframe. If no value is provided, a default value of 350 is used.

```
publishEvent({
  eventType: Constants.SHARED_EVENT_TYPE.LOGOUT_RESULT,
  payload: new LogoutResult({ success: true, loginFrameHeight: 350 })
})
```

Note: Clicking **Log Out** in the Voice Call Simulator performs the same action. To learn more, see Test Your Implementation with the Voice Call Simulator on page 34.

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference: logout

CHAPTER 8 Customize Your Implementation

In this chapter ...

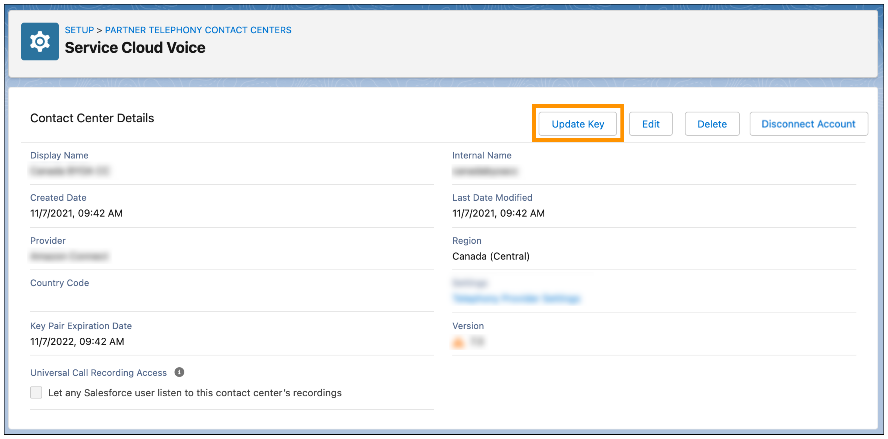
Use this information to customize the setup and add vendor value to your implementation.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Automated Key Provisioning with Service Cloud Voice for Partner Telephony

When a contact center is created and connected with the partner telephony system, the public key for the contact center can be provisioned automatically with an Apex call to the partner system.

The key can also be renewed by clicking the **Update Key** button in the contact center details page, or by using the **Update Key** dropdown action in the contact center list page.



SETUP > PARTNER TELEPHONY CONTACT CENTERS

Service Cloud Voice

Contact Center Details

Display Name
[redacted]

Internal Name
[redacted]

Created Date
11/7/2021, 09:42 AM

Last Date Modified
11/7/2021, 09:42 AM

Provider
[redacted]

Region
Canada (Central)

Country Code

[redacted]

Key Pair Expiration Date
11/7/2022, 09:42 AM

Version
[redacted]

Universal Call Recording Access

□ Let any Salesforce user listen to this contact center's recordings

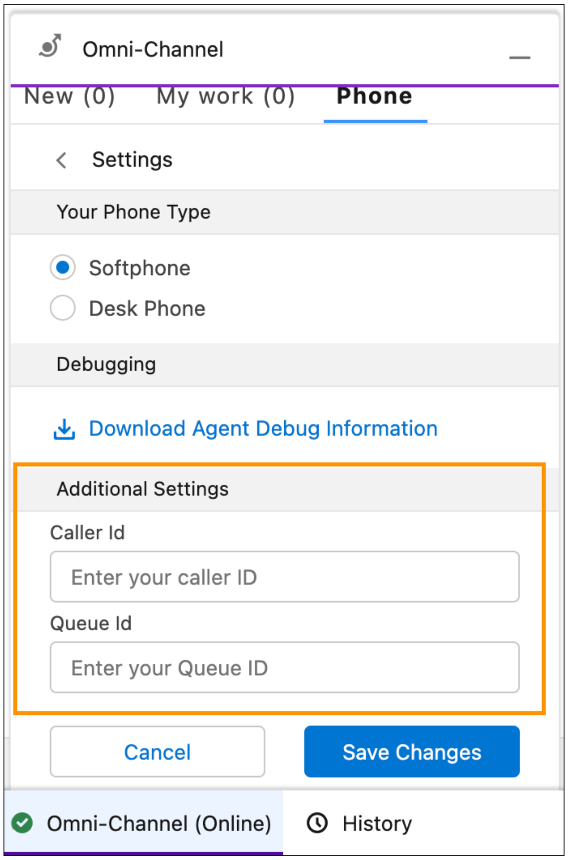
For automated key provisioning and renewal, the following requirements must be met:

1. Partners must set the `CapabilitiesSupportsKeyProvisioning` capability to `true` in their ConversationVendorInfo object.
2. The Apex integration class must implement the `KeyProvider` interface. This interface includes the `getPublicKey` method, which is called when the key is provisioned or renewed and returns the public key and its expiration date.

   When a contact center has an expired public key, the expiration date on the contact center details page and the contact center list page has a red circle with a slash icon next to it. When the public key expires within 5 days, it has a yellow warning icon next to it.

Add a Partner Settings UI to Omni-Channel

Partners can launch a Lightning component from the Omni-Channel widget with their own additional settings. This Lightning component is shown in the Additional Settings section along with other common rep settings.



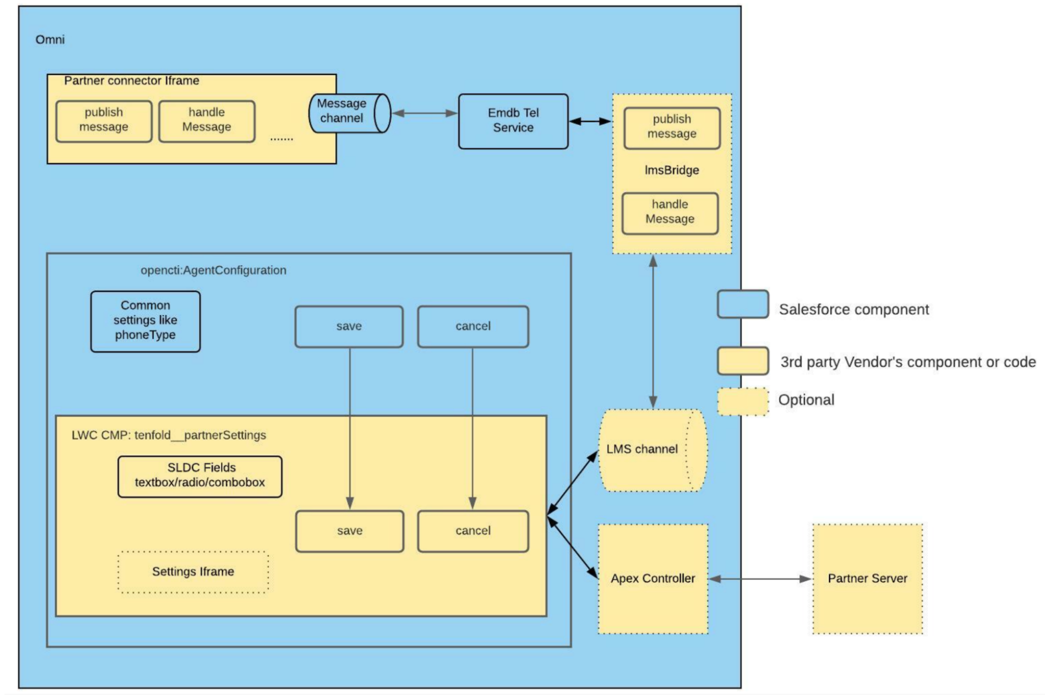
Your Phone Type

● Softphone
○ Desk Phone

This Lightning web component (LWC) can show any available LWC elements, such as radio boxes, checkboxes, comboboxes, input boxes.

To make this feature available, Partners have to provide the fully qualified name of the Lightning component from within Conversation Vendor Info, along with the LWC component in the published package. Salesforce reads the component name from the Conversation Vendor Info and displays the partner component from within the Omni-Channel widget.

To fully integrate the LWC component, partners must provide the mandatory hooks for Save and Cancel. The following diagram illustrates how the LWC component interacts with other components.



In Conversation Vendor Info, set the `TelephonySettingsComponent` field to {namespace}:{componentName}.

Sample `agentConfig.html`:

```
<template>
  <div class="slds-p-left_large slds-p-right_large">
    <lightning-input type="text" label="Caller Id" onchange={onCallerIdChange}
value={callerId} placeholder="Enter your caller ID">
    </lightning-input>
   <lightning-input type="text" label="Queue Id" onchange={onQueueIdChange} value={queueId}
 placeholder="Enter your Queue ID">
    </lightning-input>
  </div>
</template>
```

Sample `agentConfig.js`:

```
import { LightningElement, track, api } from 'lwc';
export default class AgentConfig extends LightningElement {
    @track queueId;
    @track callerId;
    _settings = {};

    onQueueIdChange(event) {
        this.queueId = event.detail.value;
    }

    onCallerIdChange(event) {
        this.callerId = event.detail.value;
    }

    @api
    save() {
        this._settings = {
            queueId: this.queueId,
            callerId: this.callerId
        };
    }

    @api
    cancel() {
        this.queueId = this._settings.queueId;
        this.callerId = this._settings.callerId;
    }
}
```

Customize Error Messages

Improve rep efficiency by displaying custom error messages in the Omni-Channel utility whenever a telephony action fails. By default, generic labels are displayed.

Custom error messages can be displayed in response to failed telephony actions. For example, you can display a custom error message in response to a failed `acceptCall` action. Here's an example of how the custom error message flow works when the `acceptCall` telephony action fails.

1. The rep tries to accept the call.
2. Salesforce passes this information to the Connector API (`vendorConnector`), which in turn passes the information to your telephony provider.
3. The Connector API responds.
4. If the error is of type `CustomError`, Salesforce displays a custom error message. Otherwise, the default error string for `MESSAGE_TYPE.ACCEPT_CALL` is displayed.

Here's a sample, taken from the baseConnector.js file in GitHub.

```
} catch (e) {
  isSupervisorConnected = false;
  if (e instanceof CustomError) {
    dispatchCustomError(e, constants.MESSAGE_TYPE.ACCEPT_CALL);
  } else {
    dispatchInfo(constants.INFO_TYPE.CAN_NOT_ACCEPT_THE_CALL, {messagetype:
constants.MESSAGE_TYPE.ACCEPT_CALL, additionalInfo: e});
  }
}
```

Custom error messages can also be displayed at any time based on the `publishError` event type (`eventType`). Here's an example of how the custom error message flow works with `publishError` `MUTE_TOGGLE` event type.

1. The rep mutes the call.
2. Salesforce passes this information to the Connector API (`vendorConnector`), which in turn passes the information to your telephony provider.
3. The Connector API responds.
4. If the error is of type `CustomError`, Salesforce displays a custom error message. Otherwise, the default error string for `ERROR_TYPE.CAN_NOT_MUTE_ALL` is displayed.

Here's a sample, taken from the baseConnector.js file in GitHub.

```
} catch (e) {
  if (e instanceof CustomError) {
    dispatchCustomError(e, constants.MESSAGE_TYPE.MUTE);
  } else {
   dispatchError(constants.ERROR_TYPE.CAN_NOT_MUTE_CALL, e, constants.MESSAGE_TYPE.MUTE);

  }
}
```

After you create a custom error message, whenever you invoke an API, a promise object is returned. If the promise resolves to a failure, Salesforce looks for a CustomError object to the promise response. If a CustomError object exists in the response, Salesforce displays the custom error. Otherwise, the default error message is displayed.

Note: If a CustomError object exists in the promise response but Salesforce can't find the labelName, the following message appears, and Salesforce doesn't fall back to the generic error message, "We couldn't find [namespace.customErrorLabel]."

Whenever you create a custom error message, perform a test to ensure the message appears when the appropriate telephony action or event fails. Perform the test through the Errors section of your demo connector phone simulator. You can find out more about the Errors section in the remote.html file of the demo connector in GitHub:

```
<div class="slds-checkbox">
  <input type="checkbox" name="options" id="throwErrorCheckbox"></input>
  <label class="slds-checkbox__label" for="throwErrorCheckbox">
  <span class="slds-checkbox_faux"></span>
  <span class="slds-form-element__label">Display an error</span>
  </label>
</div>
```

Create Custom Error Messages

There are two ways to create custom error messages:

- Create the custom labels for the error messages through the UI by following the steps in the Custom Labels page.
- Create the custom labels for the error messages yourself, distribute them in a managed package to AppExchange, and finally deploy the managed package in your org. See the `CustomError` Connector API object for more information.

After you create a custom error message, perform a test to ensure the message appears when the appropriate telephony action or event fails.

Test Custom Error Messages

You can only test one custom error message at a time. Repeat these steps for each custom error message you want to test.

To test the custom error messages you configured:

1. In the Errors section of your demo connector phone simulator, select **Display on error** to display the message during the simulation.
2. In the text box, enter the custom label you want to test. The custom label must be in the format `<namespace>.<labelName>`, where namespace is the name of the Salesforce org's namespace prefix, and labelName is the name of the custom label in your org. If you don't have a namespace, set the value to `c`. For example, `c.cannotMuteLabel`.
3. Simulate the telephony action or event in the Omni-Channel utility and view the error message to make sure the customized version appears.

4. Deselect **Display or error** after you're done testing your custom error messages.

SEE ALSO:

Salesforce Help: Custom Labels

Service Cloud Connector API Reference: publishError

Service Cloud Connector API Reference: customError

Create and Deploy Your Package

Test Your Implementation with the Voice Call Simulator

Communicating with Lightning Components

If you set up Service Cloud Voice with Partner Telephony, you must enable communication between the Lightning components and your telephony system. You can do this by creating a Lightning Message Service (LMS) bridge and/or configuring the Service Cloud Voice Toolkit APIs.

The Lightning Message Service bridge lets you:

- Send messages to the connector and receive messages from the connector at any time.
- Send custom messages between the telephony system and components on the page.

The Service Cloud Voice Toolkit API lets you:

- Receive telephony events, including start, end, and mute call events.
- Take advantage of a predefined set of Voice-related messages, such as CALL_STARTED, CALL_CONNECTED, and PAUSE_RECORDING.

We recommend using the LMS Bridge for communication between the Omni-Channel softphone and your components because it's more secure and flexible. However, you may choose to use the Service Cloud Voice Toolkit API instead because of its simplicity.

Use the Lightning Message Service Bridge

Use the Lightning Message Service Bridge component to enable communication between the telephony system and other Lightning components.

Use the Service Cloud Voice Toolkit API

Configure the Service Cloud Voice Toolkit APIs to listen to telephony events and perform telephony actions.

Use the Lightning Message Service Bridge

Use the Lightning Message Service Bridge component to enable communication between the telephony system and other Lightning components.

To use the Lightning Message Service (LMS) channel in your contact center, create a Lightning component using Aura or a Lightning Web Component that serves as the bridge component. This component supports communication between the connector and other Lightning components on the page.

Using Aura

- It must implement the Aura interface `service_cloud_voice:messageBridge`.
- It must reference the LMS channel you defined with namespace.
- It must implement the `publishMessage` method to publish connector messages to the channel.

- It must call the handleMessage Action handler to send messages to the connector.
- It must not have any UI markups.

.cmp example:

```
<aura:component implements="service_cloud_voice:messageBridge">
  <lightning:messageChannel type="{your_namespace}__{LMS_channel_name}__c"
aura:id="sampleMessageChannel" onMessage="{!c.handleMessage}"/>
</aura:component>
```

Controller.js sample:

```
({
    publishMessage : function(component, event, helper) {
        var message = event.getParam('arguments').message;
        component.find('sampleMessageChannel').publish({ source : 'CONNECTOR',
payload:message});
    },

    handleMessage: function(cmp, message, helper) {
        var messageWrapper = message.getParams();
        if(messageWrapper.source != 'CONNECTOR') {
            cmp.get("v.handleMessage")(messageWrapper.payload);
        }
    }
})
```

Using Lightning Web Components (LWC)

- There's no interface equivalent in LWC.
- It must reference the LMS channel you defined without a namespace.
- It must expose the `publishMessage` API to publish messages to the channel.
- It must expose a handleMessage attribute and use that as a callback to send messages to the connector.
- It must not have any UI markups.

.js example:

```
import { LightningElement, track, wire, api } from 'lwc';
import { publish, subscribe, APPLICATION_SCOPE, MessageContext } from
'lightning/messageService';
import SAMPLEMC from "@salesforce/messageChannel/{LMS_channel_name}__c";

export default class Lms_LWC extends LightningElement {
    @wire(MessageContext)
    messageContext;

    @api handleMessage;

    constructor() {
        super();
    }

    connectedCallback() {
        subscribe(this.messageContext,
            SAMPLEMC,
            (message) => {
                if(message.source != 'CONNECTOR'){
                    this.handleMessage(message.payload);
                }
            },
            {scope: APPLICATION_SCOPE}
        );
    }

    @api
    publishMessage(message) {
       publish(this.messageContext, SAMPLEMC,  { source : 'CONNECTOR', payload:message});

    }
}
```

Note:

- To avoid echo, a source property is inserted during message publishing. When a message from the connector is published to the channel, it's filtered out and not sent back. The source property may not be needed if the communication is one-way.
- The connector starts receiving messages sent from the message bridge after the connector is fully initialized.

Configure the Contact Center to Use the Message Bridge Component

After creating the component, configure the contact center to use it.

Using the ConversationVendorInfo setup entity (for Production): In a production environment, specify the message bridge component's fully qualified name in the ConversationVendorInfo record from the same managed package that has the actual Lightning component. When a Salesforce admin imports an XML file to create a contact center, it references the ConversationVendorInfo to read the message bridge component name and configure the contact center custom settings automatically.

Using CallCenter Metadata API (for Local Testing): Add the message bridge component FQN to the contact center's custom settings for testing. This can be done using theCallCenter Metadata type. Set the **messageBridgeComponentFqn** value in the Metadata file to the fully qualified name of your bridge component. For an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<CallCenter xmlns="http://soap.sforce.com/2006/04/metadata">
...

[illegible]
...
</CallCenter>
```

Test Your Bridge Component

You can test the bridge component using the Messages section of the Voice Call Simulator. To learn more, see Test Your Implementation with the Voice Call Simulator on page 34.

Limitations

See Lightning Message Service Limitations.

Use the Service Cloud Voice Toolkit API

Configure the Service Cloud Voice Toolkit APIs to listen to telephony events and perform telephony actions.

To learn more about the Service Cloud Voice Toolkit API, see the Toolkit API documentation in the Service Cloud Voice Implementation Guide. The following examples in GitHub show a Lightning Web Component and an Aura component that can subscribe to and handle call events (such as CALL_STARTED, CALL_CONNECTED).

Note: Components (such as the Lightning Message Bridge) that implement the `lightning:backgroundUtilityItem` interface aren't supported locations for the Service Cloud Voice Toolkit API. Use this API in a component that resides on a page, or resides in a visible utility bar component.

SEE ALSO:

Service Cloud Voice Implementation Guide: Toolkit API

CHAPTER 9 Start Calls

In this chapter ...

This section provides guidelines related to starting calls.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

SEE ALSO:

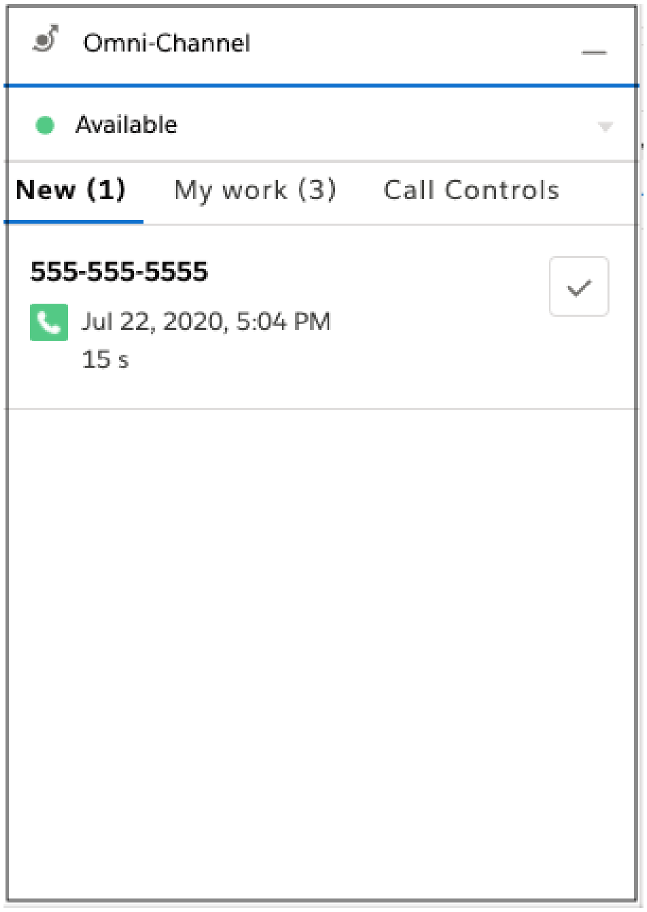
Call Scenario Diagrams

Accept Inbound Calls in Omni-Channel

Now that you've added support for inbound calls, open the Voice Call Simulator and click New Inbound Call.

For details, see Test Your Implementation with the Voice Call Simulator on page 34.

The call appears in the Omni-Channel utility.

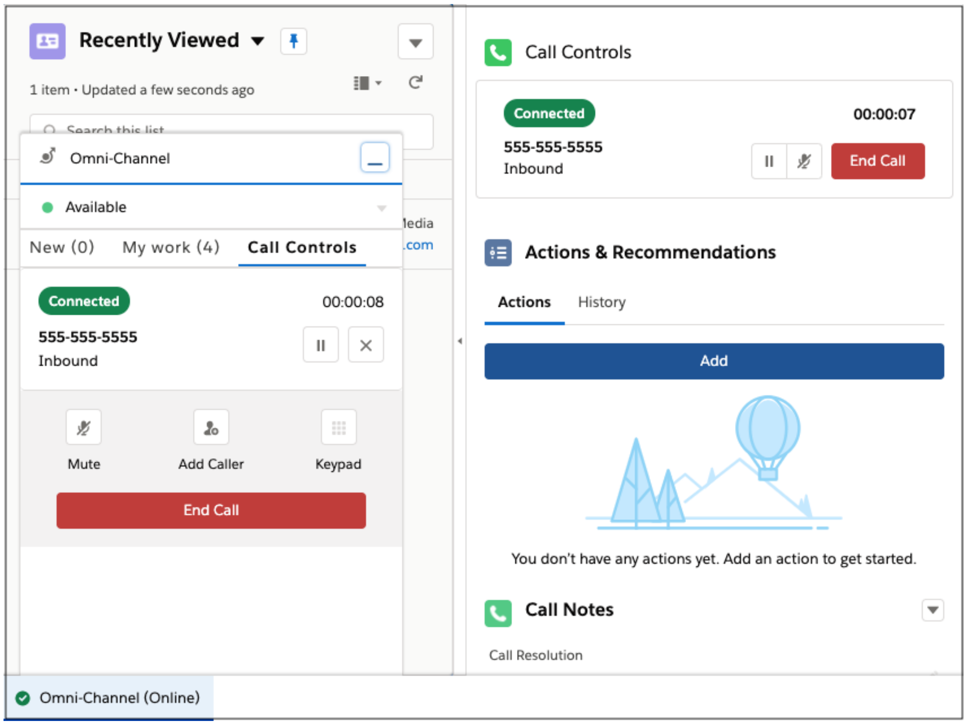


555-555-5555

Jul 22, 2020, 5:04 PM

15 s

To accept the call, click the checkmark. You can mute, unmute, transfer, and end the call. Salesforce opens the VoiceCall record that is created.



Integrate Incoming Voice Call Creation

For inbound calls, when the telephony system receives the incoming call and prepares to route the call to a rep, the telephony system can invoke Service Cloud Voice REST APIs to create the essential Voice Call record to represent the conversation.

You can use following REST endpoint: `https://{org_domain_name}.my.salesforce-scrt.com/telephony/v1`. For `org_domain_name`, specify the value as defined in Set Up Single Sign-On on page 40.

The Service Cloud Voice REST APIs use JWT for authorization.

- Salesforce requires that a JWT is signed using RSA SHA256, which uses an uploaded certificate as the signing secret.
- As part of your contact center setup, ensure that the call center has been updated with the certificate you generated in Generate a Self-Signed Certificate with OpenSSL on page 16. The generated private key is used to sign the JWT bearer token payload.
- Store the private key so you can retrieve it whenever you generate a JWT token.
- The public key is saved in the call center information, whereas the private key is saved on the telephony provider side to sign the JWT bearer token. Ideally, the private key is never exposed on the network and stays on the server where it was generated.

For instructions on setting up authorization, see Telephony Integration REST API Authorization.

Once you have authorization set up, call the Create VoiceCall REST API. Use the `callAttributes` parameter to ingest Interactive Voice Response (IVR) data into a VoiceCall record.

SEE ALSO:

Service Cloud Voice Implementation Guide: Telephony Integration REST API Authorization

Service Cloud Voice Implementation Guide: Create VoiceCall

Record Linking

Use record linking to associate a voice call with other related records.

Related Records

Add the Voice Call related list to a Lightning record page to show a list of calls associated with a record. For example, add the list to your Case record page, so reps can see a case's associated calls and learn about the customer's interactions with your company.

1. From the Object Manager in Setup, select the object that you want to add the Voice Call related list to. For example, to add the Voice Call related list to a Case page layout, select **Case**.
2. Select **Lightning Record Pages** .
3. Select the page that you want to modify, then click **Edit**.
4. Drag the Related List - Single component to the page layout.
5. In the Related List menu, select **Voice Calls**.
6. Save your changes.

Object Linking

When a rep accepts a call in the workspace, prompt them to choose from recommended records, search for a record, or add a new one.

1. From Setup, enter Channel-Object in the Quick Find box, then select **Channel-Object Linking**.
2. Click **New Linking Rule**.
3. Select the Phone channel and the object to link to (such as Contact).
4. Create a rule name and description.
5. Set rule actions for Action for No Record Found and Action for Single Record Found. For example, if no matches are found for a contact, include a rule to automatically create and link a record, or prompt the rep to search for or create a record. If a single matching record is found, set a rule to automatically link the record. Alternatively, prompt the agrepnt to pick the suggested record, search for a record, or create a record.
6. Save your work.
7. After completing the Phone setup, add the Object-Linking Notifications background utility in the App Manager. This utility displays toast messages in the console that prompt the rep to link a suggested record or add a new one.
8. From Setup, enter *App Manager* in the Quick Find box, then select **App Manager**.
9. Click **Edit** in the action menu for your app.
10. Click **Utility Items (Desktop Only)** | **Add Utility Items** | **Object-Linking Notifications**.
11. Save your work.

Using Flow for Record Linking

Flow builder can also be used to link a case or contact to a voice call.

1. From Setup, enter *Flow* in the Quick Find box, then select **Flows** from Process Automations.

2. Use the logic and data elements from the flow builder to trigger events such as opening cases associated with a voice call or opening a contact associated with a voice call.

SEE ALSO:

Salesforce Help: Channel-Object Linking

Queued Callbacks

When a customer makes an inbound call, the telephony system first creates a voice call and then routes the call to the available rep. If no rep is available and the customer requests a callback, you must publish a QUEUED_CALLBACK_STARTED event.

Call the Service Cloud Connector API `publishEvent()` method with a Call payload having `callType` of `callback` and an original inbound call ID as `initialCallId` to link it with the new callback voice call.

On the QUEUED_CALLBACK_STARTED event, a new conversation and Voice Call record are created and the AgentWork is created.

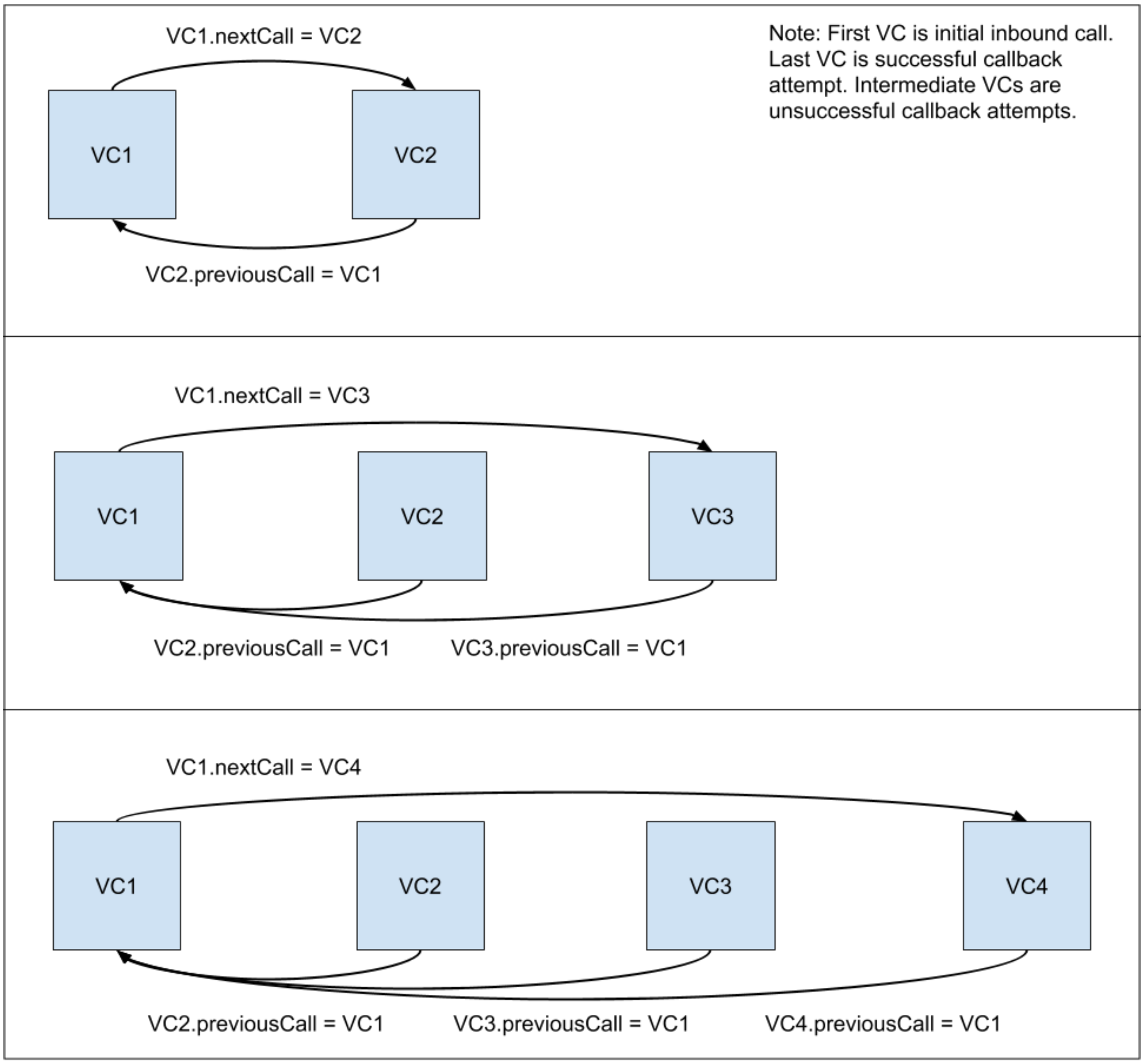
```
// Create phone call object
const call = new PhoneCall({
            callId: vendorCallKey,
            phoneNumber : '{CallBack number}',
            callInfo : { initialCallId: previousVendorCallKey},
            callType: Constants.CALL_TYPE.CALLBACK.toLowerCase()/*'callback'*/,
            contact: new Contact({ phoneNumber }),
            callAttributes: { participantType: Constants.PARTICIPANT_TYPE.INITIAL_CALLER
} });

// Publish the event
publishEvent({ eventType: Constants.VOICE_EVENT_TYPE.QUEUED_CALL_STARTED, payload: new
CallResult({ call })});
```

Note: initialCallId is not a mandatory field. If you do not have a initial inbound call before a callback request, you can skip it.

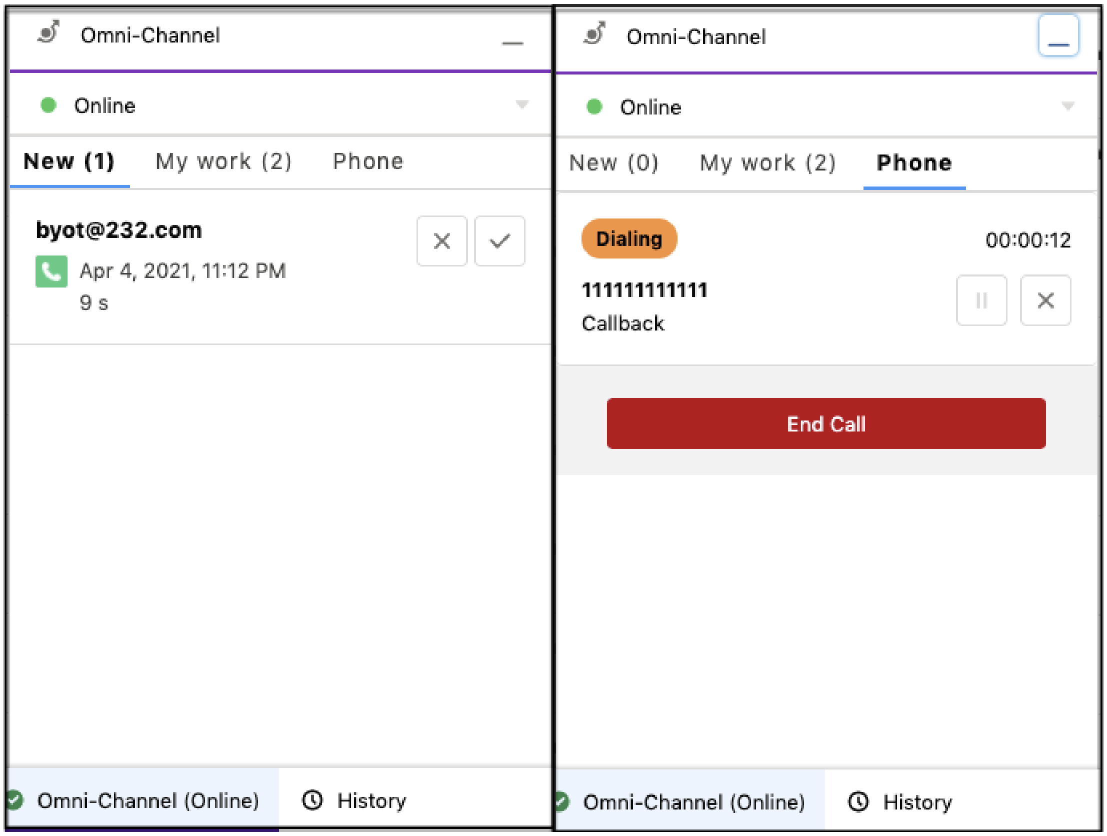
For example, suppose that a customer makes an inbound call and the connecter creates a VoiceCall VC1. If no reps are available, the IVR asks the customer if they want a callback. If a customer asks for a callback, then the telephony system can queue a callback after an initial delay (in which the callback is initiated as soon as a rep is available) or a callback is scheduled for a customer on a particular date.

A callback can also be requested from a particular rep. The telephony system creates a new voice call VC2 and the connector sends a QUEUED_CALL_STARTED event as per rep availability and routing rules along with callback phone number and VC1 as the `initialCallId`. The callBack is displayed in Omni-Channel as shown in the following screenshot. When a rep accepts the call, an outbound call is triggered to the given phone number and the `previousCall` field on the VC2 becomes VC1 and next call on VC1 becomes VC2 in the Voice Call details.

Here are some scenarios where a callback request is accepted but fails to connect with the customer.



Callback acceptance and outbound call dialing:



If a rep declines the call or the call times out, the telephony system can try to connect to another available rep with the same VC2.

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference: publishEvent

Service Cloud Connector API Reference: CallResult

Service Cloud Connector API Reference: PhoneCall

Let Reps Control the Callback Experience

Let Reps Control the Callback Experience

Customize the way reps handle callbacks.

By default with queued callbacks, when a rep accepts a callback request, the Omni-Channel utility automatically dials the callback number and the call is immediately active.

To give reps more control, configure the callback feature so reps can decide how the call is handled when they accept a callback request. Reps can view the callback details with information about the customer's request. Then they can transfer the callback request to another rep, initiate a callback through click-to-dial with the listed preferred phone number, or contact the end user at another phone number.

Available in API version 60.0 and later for Service Cloud Voice with Partner Telephony.

Prerequisites:

Enable the option so that customers can create callback requests.

To allow click-to-dial callbacks or transfers from Contact Requests, follow these steps.

1. When a customer submits a contact request to request a callback, the partner creates a ContactRequest record and an associated AgentWork record. The ContactRequest has the IsCallback field set to `true`.
2. A rep gets assigned the agentWork record through the Omni-Channel widget and accepts the work.
3. The rep can review the details of the contact request and determine their next steps.
   - If the rep is ready, they call the customer by using click-to-dial from one of the options on the Contact Request page. This callback is handled through the ContactRequest object, and it creates an additional agentWork record. To ensure that click-to-dial calls made through the content request are marked as callbacks rather than outbound calls, verify that `ContentRequest.IsCallback` is `true`.
   - If the rep prefers to transfer the contact request and its associated agentWork instead, the rep clicks **Transfer** on the ContactRequest record page. The partner must send Salesforce the `TRANSFER_CALLBACK_REQUEST` Connector event to open the Phone transfer view in the Omni-Channel utility. When the rep identifies who they want to transfer the call to and selects **Transfer**, Salesforce sends an `add_participant` event to the partner with `isBlindTransfer=true`.
4. The partner listens for the AgentWorkClosed event to release the rep's capacity so that they can receive the next call.

SEE ALSO:

Queued Callbacks

Object Reference for the Salesforce Platform: ContactRequest

Salesforce Help: Return a Callback Request

Outbound Dialers with Service Cloud Voice for Partner Telephony

You can use a preview dialer or a progressive dialer with Service Cloud Voice for Partner Telephony.

Preview (Push) Dialer

Note: The PREVIEW_CALL_STARTED Connector API call is no longer supported starting with the Winter '23 release.

To use a preview dialer, create a preview dialer Lightning component and then use the `startPreviewCall` method in the Aura-based Toolkit API to initiate outbound calls. Calling this method ensures that the VoiceCall record has the **CallType** field set to `Outbound` and the **CallOrigin** field set to `Preview`. See the Sample Aura Component in GitHub.

Progressive Dialer

A progressive dialer works similarly to inbound calls. The vendor telephony system dials the customer without notifying the rep. After the customer picks up the call, the vendor follows the inbound call flow (that is, create a Voice Call, send CALL_STARTED, accept or decline).

Enable the Phone Book for Outbound Calls

Enable the phone book so reps can use their speed-dial list to make rep-to-rep and rep-to-queue calls. When the phone book is enabled, a rep can view a list of reps and queues in the Omni-Channel utility and place a call to the destination rep. For rep-to-queue calls, a rep selects the queue from the phone book, which determines which rep to contact.

This feature leverages the Service Cloud Connector API getPhoneContacts() function to get the list of phone contacts and contact types that appear in the Omni-Channel utility.

This feature applies to the following telephony model:

- Service Cloud Voice with Partner Telephony

Enable the Phone Book

Perform the following steps using the Connector API to enable the phone book feature.

1. When implementing `getCapabilities()`, set the value of `hasPhoneBook` in `CapabilitiesResult` to `true`.
2. When implementing `dial()`, set the appropriate `callType` for the agent (rep) and queue contact types. For example, if contact type is 'agent', set `callType` to InternalCall, which represents a call made between reps within a contact center. In our demo implementation of `dial()`, contact type 'agent' is set to `callType InternalCall`, and contact type 'queue' is set to `callType Outbound`.
3. To ensure `internalCall` works seamlessly, make sure the Salesforce administrator configures support for rep-to-rep calls at the org level by selecting the **Voice Call (VoiceCall)** Change Data Capture entity.
4. When a rep makes a phone book InternalCall, the rep's `participantType` should be `Initial_Caller`, and the receiving rep's `participantType` should be `Agent` in your `CallResult`. See the demo implementation of `dial()` for an example.
5. To ensure the reps' respective names appear in voice call transcripts, make sure the participantId appears in the transcripts. See the Create a Transcript API for more information.
6. When two reps share the same voice call record for `callType InternalCall`, the first rep (Rep 1) is a standard user with restricted access to the VoiceCall object, while the second rep (Rep 2) becomes the owner of the VoiceCall record. This might cause Rep 1 to lose access to the VoiceCall record. To ensure Rep 1 can access the VoiceCall record, create sharing rules based on group membership by adding all reps who use the phone book feature as members of a Salesforce group that has "Read Only" access created for that group.

Test the Phone Book

1. In the **Capabilities** section of your Service Cloud Voice (demo connector) phone simulator, select **Support Phone Book**. You can also find the `hasPhoneBook` capability listed in the baseConnector.js demo connector file in GithHub.
2. In the Omni-Channel utility, click the phone book icon (person avatar) just to the right of the phone number field.
3. Verify the directory of rep and queue contacts appears.

Note: If two reps are on a phone book `InternalCall` and one of the reps transfers the call to a third rep, the new VoiceCall record created for the third rep will be callType `Transfer`.

SEE ALSO:

Service Cloud Connector API Reference

Salesforce Help Configure the Phone Book for Outbound Calls

Set the Voice Call Record Type

Customize the page layout of a voice call record.

The record type determines the page layout of a voice call record. Configure your system to automatically set the record type when you create the voice call record by passing in `“RecordTypeId”:”RECORD_TYPE”` inside the `callAttributes` parameter, where `RECORD_TYPE` is the unique identifier for the voice call record type. For example, `{"callAttributes": "{\"RecordTypeId\": \"012300000012BYNQAG\"}"}`.

After a voice call record is set, reps can manually change its record type through the Salesforce Lightning Experience UI.

SEE ALSO:

Service Cloud Voice Implementation Guide: Create a Voice Call Record

Salesforce Object Reference for the Salesforce Platform: VoiceCall

Send Voicemails to Reps

Let customers send your reps voicemails that reps can review along with a transcription.

When your customer chooses to leave a voicemail through your telephony system, use the Telephony API to create or update a VoiceCall record. When calling the API, set the `callOrigin` parameter to `Voicemail`. See Create a Voice Call Record and Update a Voice Call Record in the Telephony API documentation. This call record can contain both the recording and the transcription.

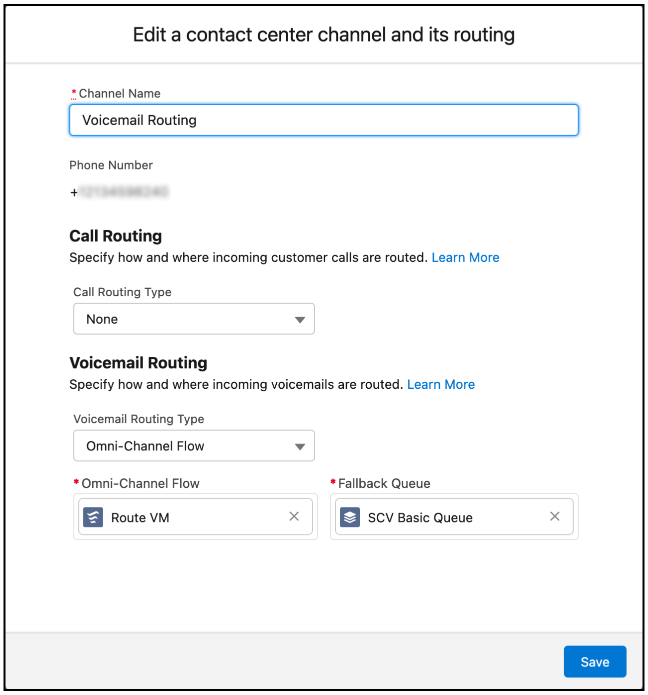
Note: When a voicemail is routed to a rep, if the transcription is still being processed in Salesforce, it may not show up when the rep accepts the voicemail. If that’s the case, the rep can refresh the page to see the transcription when it’s ready.

Setup for Routing Voicemails

Voicemails are routed through Omni-Channel using an Omni-Channel flow. An admin can create a flow that routes voicemails to a particular queue. You can create a queue in setup and add a routing configuration to the queue (by using a routing other than External Routing).

You can also define a single flow that routes both voice calls and voicemails, and then branch your routing by using the `callOrigin` field of the VoiceCall record.

Voicemail routing can be defined for any inbound phone number in the contact center channels section.



Edit a contact center channel and its routing

* Channel Name

Voicemail Routing

Phone Number

+[redacted]

Call Routing

Specify how and where incoming customer calls are routed. Learn More

Call Routing Type

None

Voicemail Routing

Specify how and where incoming voicemails are routed. Learn More

Voicemail Routing Type

Omni-Channel Flow

* Omni-Channel Flow

Route VM

* Fallback Queue

SCV Basic Queue

Sample Sequence of Operations

1. The customer calls the Contact Center number.
2. The partner system creates a VoiceCall using the Create a Voice Call Record Telephony API.
3. If no reps are available, the customer chooses to leave a voicemail.
4. The customer records a voice message.
5. The partner system updates the VoiceCall using the Update a Voice Call Record Telephony API. With this API call, `isActiveCall` is set to `true` to ensure that the conversation isn't closed, as we have more updates ahead. Also, `callOrigin` is set to `"Voicemail"` to ensure that the call is tagged as a voicemail message. (Alternatively, the partner can set the `callOrigin` when creating the call earlier if they know this information when the call record is first created.)
6. The partner system creates a transcription using the Create a Transcript Telephony API.
7. The partner system executes an Omni-Channel Flow using the Telephony API. With this API, the `dialedNumber` parameter contains the routing configuration that is defined in the Contact Center details page. (Make sure that `callOrigin` is updated to `"Voicemail"` before this API call so that the flow picks the voicemail routing instructions.)
8. The partner system does a final update (using the Update a Voice Call Record Telephony API) so that the conversation is completed by setting `isActiveCall to false`, along with the recording details.

Example Code

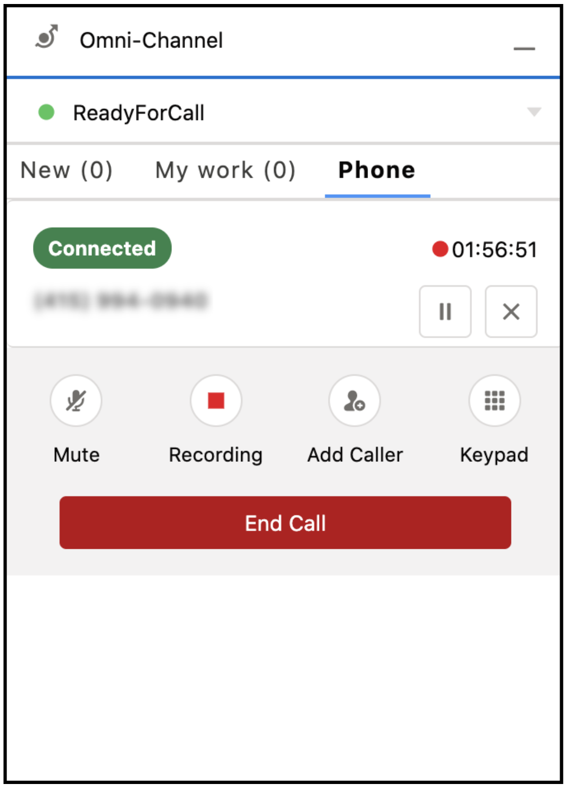
For a complete example implementation, see the demo connector code.

A sample REST call to the Telephony API that includes the `callOrigin` parameter when creating a VoiceCall record:

```
POST /telephony/v1/voiceCalls

{
  "callCenterApiName": "MyContactCenter",
  "vendorCallKey": "5324881f-1e84-4367-8930-f69a74b30fff",
  "to": "999999999",
  "from": "999999999",
  "initiationMethod": "Inbound",
  "callOrigin": "Voicemail",
  "startTime": "2022-08-02T17:32:28Z",
  "participants": [
    {
      "participantKey": "999999999",
      "type" : "END_USER"
    }
  ]
}
```

Hide Call Controls

You can hide many of the call control buttons that appear to the rep in the Omni-Channel widget. These buttons can be hidden or shown for each call. By default, buttons are visible.



These buttons can be hidden from view.

- Mute
- Record (or Recording)
- Add Caller
- Blind Transfer
- Merge

- Swap

The keypad is always displayed.

To hide a button, specify `false` for the show method in the `CallInfo` object. For instance, use `showAddCallerButton` to hide or show the add caller button. The `CallInfo` object is used by the `PhoneCall` object for methods such as `acceptCall`.

SEE ALSO:

Service Cloud Connector API Reference

Manage Multiparty Calls and Consult Calls

Include up to six participants on a voice call, including the rep and customer. In addition to the six participants, a supervisor can listen in or barge in. While on a call with a customer, a rep can make a separate consult call before merging the calls together or ending the consult call. During the consult call, multiparty call participants are on a type of hold. Telephony partners specify whether to disable hold music so that call group participants can continue their conversation.

Before you begin, get access to the demo connector sample code in GitHub.

A product manager (PM) must set these org permissions. If they're set, the feature is automatically enabled and requires no admin setup.

- Service Cloud Voice: Multi-party conferencing
- Service Cloud Voice: Multi-party consult

Multiparty Calls and Consult Calls Limitations and Considerations

Keep these limitations and considerations in mind when setting up and using multiparty and consult calls.

Feature compatibility and support:

- Blind transfers aren't supported for multiparty calls.
- Browser refresh isn't fully supported during multiparty calls.
- Service Cloud Voice Toolkit API isn't supported for multiparty calls.
- Omni-Channel Unified Routing (pilot) isn't compatible with multiparty calls.
- Compact layout call controls on the Voice Call record page aren't compatible with multiparty calls.
- When Service Cloud Voice is in Voice Resiliency on page 30 mode, consult calls aren't supported. A rep that has been requested to join a consult call won't receive the call.

User experience:

- A rep on a multiparty call can't end the call for all participants at one time. They must end the call for each participant one by one, or each participant must leave the call individually.
- The contact list for adding a participant to a multiparty call doesn't filter out reps that are already part of the call group. If a telephony partner requires this behavior, they are responsible for handling it when returning the getContacts()or getPhoneContacts() API.
- Transfer calls and multiparty calls are not automatically merged if a rep selects New Call.
- There are no hold and resume options on the call entry for multiparty or consult calls.
- If the multiparty calls feature is enabled, phone controls in the Omni-Channel utility are updated for standard calls and multiparty calls.
- When the call group is toggled open, the timer defaults to 00:00 for less than a second before displaying the correct time.

- In the demo connector implementation, if a rep adds an external contact to a multiparty call, the external contact appears in the call group for only the rep who added them. The external contact doesn't appear in the call group for other call participants.
- A rep can start a consult call when there is an active multiparty call.
- A rep who receives a consult call isn't able to add participants to the consult call or transfer the consult call. Their role is limited to consulting until the consult call is merged with a multiparty call.
- A rep can't end a multiparty call while on an active consult call. First, the rep should end or merge the consult call, then end the multiparty call.
- A call participant's name may appear incorrectly after starting a consult call or when merging a consult call and multiparty call.
- If the multiparty calls feature is enabled, a supervisor with a Standard User profile can't pause or resume call recording. If the supervisor joins an active call to listen in, reps on the call don't get a supervisor joined notification.

Add Participants to a Multiparty Call

1. To add a new participant to an ongoing call, use `addParticipant()`. Check whether `callAttributes` for the `PhoneCall` include `isAutoMergeOn`. The `isAutoMergeOn` attribute signifies that the call created with `addParticipant()` should be merged into the ongoing call without putting the initial caller on hold.
2. As each new participant is added, call `addParticipant()`.
3. When implementing `addParticipant()`, create a Voice Call of `CallType Transfer`, and use these call attributes to auto merge the new participant to an existing multiparty call when the rep clicks the **Add User** button.

For an example, see the `vendor-sdk.js` file in the sample demo connector.

The `isMultipleParticipants()` method in the phone call controls template checks the length of `phoneCallList` and shows a call group if the length is greater than 1. The call group appears if there are additional participants in the call, aside from the rep and customer.

Mute and Unmute Participants in a Multiparty Call

Mute call participants individually. Muting all participants in the call with one button click isn't supported for multiparty calls.

1. Pass a call object as a parameter to `mute()` to mute a specific call participant.
2. Optionally, pass `isGlobal` to `MuteToggleResult` to specify whether the mute action is for the rep (global mute) or for another participant (entry mute). If `isGlobal` is `true`, the rep is muted. If `isGlobal` is `false`, a participant in the call group is muted. The default is `false`.

```
mute(call) {
    const isMuted = true;
    call = this.updateCallInfo({ isMuted }, call);
    const isGlobal = /* implement condition check */
    return this.executeAsync("mute", new MuteToggleResult({ isMuted,  call, isGlobal
}));
}
```

3. To unmute a specific participant, pass a call object as a parameter to `unmute()`. Pass `isGlobal` to `MuteToggleResult`.

```
unmute(call) {
      const isMuted = false;
      call = this.updateCallInfo({ isMuted}, call);
      const isGlobal = /* implement condition check */
      return this.executeAsync("mute", new MuteToggleResult({ isMuted, call, isGlobal
}));
}
```

Remove Participants from a Multiparty Call

Remove call participants individually. Removing all participants in the call with one button click isn't supported for multiparty calls.

1. To end the call for a specific participant, pass a call object as a parameter to `endCall()`.

   ```
   endCall(call, agentErrorStatus) {
         this.log("endCall", call, agentErrorStatus);
         let destroyedCalls = this.processEndCall(call, agentErrorStatus,
   Constants.HANGUP_REASON.PHONE_CALL_ENDED, true);
        return this.executeAsync("endCall", new HangupResult({ calls: destroyedCalls }));
   }
   ```

Create and Manage Consult Calls

To support consult calls, make these adjustments to your connector code.

1. Vendors should set the value of `hasConsult` in `VoiceCapabilitiesResult` to `true`. After Salesforce receives a value of `true` from the `getVoiceCapabilities()` method, the `hasConsult` capability is turned on for reps that support consult calls.
2. In your `dial()` implementation, make these adjustments.
   a. Include a `dialOptions` object as a parameter to specify whether `isConsultCall` is `true`. When a rep starts a consult call, Salesforce calls `vendorConnector.dial(contact, dialOptions)` where `dialOptions: {isCallback, isConsultCall}` with `isConsultCall` as `true`. `isConsultCall` also exists in `call.callAttributes`. Salesforce creates a new Voice Call of CallType Consult upon resolving the `dial(contact, isConsultCall)` promise.
   b. When implementing `dial(contact, isConsultCall)`, create a new Consult call leg while the rest of the multiparty call has no hold music. Partners can choose whether to play hold music in a multiparty call when a consult call is started. Hold without music allows call group participants to continue their conversation while a consult call is active. Note that the Connector API `hold()` method isn't invoked for the call group even though the UI shows a hold state for the call group.

Merge a Consult Call and Multiparty Call

1. To merge a multiparty call and a consult call together, implement `conference()` where the first call is the initial multiparty call, and the second call is the consult call, where `call.callAttributes.isConsultCall = true`.
2. Remove the participant from the consult call and add them to the original call. Close the consult call after merging the calls and return a resolved `HoldToggleResult`.

Pause and Resume Recording for Consult Calls

1. To pause recording for a consult call, pass a call object as a parameter to `pauseRecording()`. For a multiparty call, it's optional to pass a call object as a parameter when pausing recording.

2. Return a resolved `RecordingToggleResult` that includes the `callId` in `contactId`.

```
pauseRecording(call) {
   const isRecordingPaused = true;
   call = call || this.getCall({callAttributes: { participantType:
Constants.PARTICIPANT_TYPE.INITIAL_CALLER }});
   if (this.isConsultCall(call)) {
       call = this.getCall({ callAttributes: { isConsultCall : true }});
   }
   return this.executeAsync("pauseRecording", new RecordingToggleResult({
isRecordingPaused, contactId : call.callId }, this.updateCallInfo({ isRecordingPaused
}, call)));
}
```

3. To resume recording for consult call, pass a call object as a parameter to `resumeRecording()`. For a multiparty call, it's optional to pass a call object as a parameter when resuming recording.

4. Return a resolved `RecordingToggleResult` that includes the `callId` in `contactId`.

```
resumeRecording(call) {
    const isRecordingPaused = false;
    call = call || this.getCall({callAttributes: { participantType:
Constants.PARTICIPANT_TYPE.INITIAL_CALLER }});
    if (this.isConsultCall(call)) {
       call = this.getCall({ callAttributes: { isConsultCall : true }});
    }
    return this.executeAsync("resumeRecording", new RecordingToggleResult({
isRecordingPaused, contactId : call.callId }, this.updateCallInfo({ isRecordingPaused
}, call)));
}
```

CHAPTER 10 During Call Actions

In this chapter ...

This section provides guidelines about actions you can take during a call.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Associate Partner Telephony Users and Groups with Queues

Set up queue management to associate partner telephony users and groups with Service Cloud Voice queues.

Overview

We enhanced the Queue Management behavior so that it's easy for partners and their customers to associate users and groups to an associated Voice queue.

To use this feature, implement these Apex interfaces from the `service_cloud_voice` namespace.

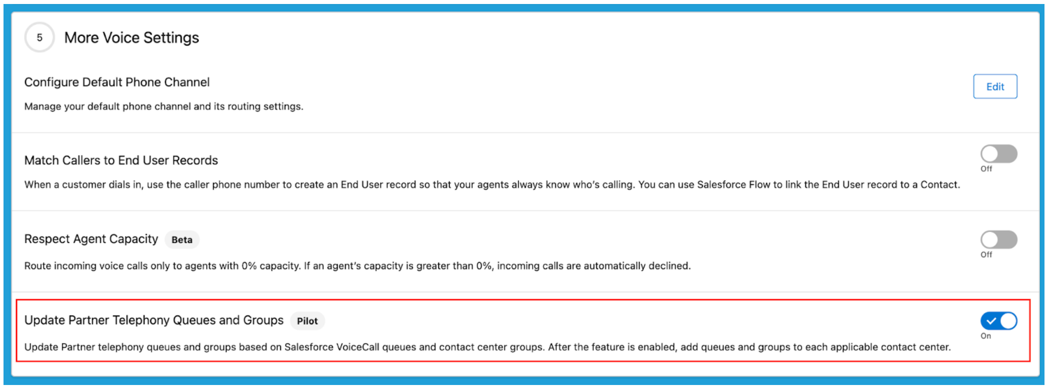
- `QueueManager`. This class describes whether Service Cloud Voice supports queue management. It contains this method.
  - `supportsQueueUserGrouping`. Indicates whether your implementation supports the queue user grouping feature.
- `QueueSetup`. This class performs all the work so that your partner telephony implementation stays in sync with Salesforce. It contains these methods.
  - `listQueues`. Lists all existing queues.
  - `createQueue`. Creates a queue.
  - `removeQueue`. Removes an existing queue.
  - `associateUsersAndGroupsWithQueue`. Associates partner telephony users and groups with a Service Cloud Voice queue.
- `GroupSetup`. This class performs all the work so that your implementation stays in sync with Salesforce. It contains the following methods.
  - `listGroups`. Lists all the existing groups.
  - `createGroup`. Creates a group.
  - `associateUsersWithGroup`. Associates users with a group.

The sample code describes how to implement these methods.

Set Up Queue Management for Partner Telephony

To enable the BYOT queue management feature, follow these instructions.

1. Contact your Salesforce representative to opt-in to this feature and turn on the org permission.
2. When your org permission is enabled, visit **Partner Telephony** in Setup and turn on **Update Partner Telephony Queues and Groups**.



5 More Voice Settings

Configure Default Phone Channel

Manage your default phone channel and its routing settings.

Match Callers to End User Records — Off

When a customer dials in, use the caller phone number to create an End User record so that your agents always know who's calling. You can use Salesforce Flow to link the End User record to a Contact.

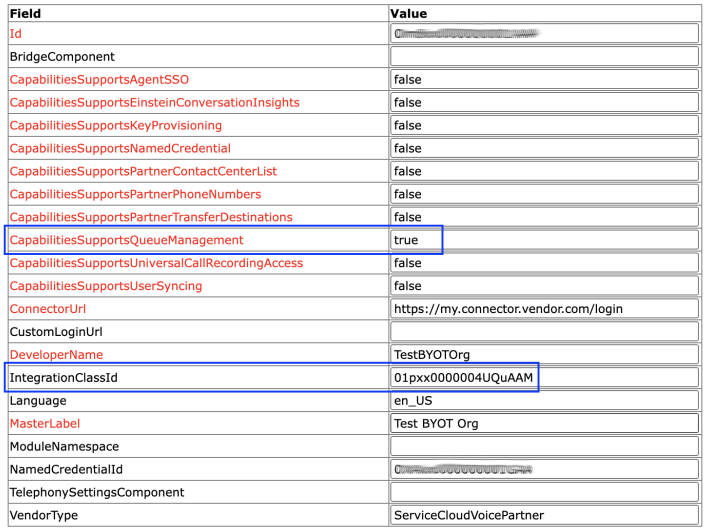
Respect Agent Capacity Beta — Off

Route incoming voice calls only to agents with 0% capacity. If an agent's capacity is greater than 0%, incoming calls are automatically declined.

Update Partner Telephony Queues and Groups Pilot — On

Update Partner telephony queues and groups based on Salesforce VoiceCall queues and contact center groups. After the feature is enabled, add queues and groups to each applicable contact center.

3. When the vendor imports the `ConversationVendorInfo` record, they must set the following fields.
   - `CapabilitiesSupportsQueueManagement`. Set this value to `true`.
   - `IntegrationClassId`. This value contains the ID of Apex implementation class. For example, `01pxx000000wxyzABC`.



Field	Value
Id	[redacted]
BridgeComponent	
CapabilitiesSupportsAgentSSO	false
CapabilitiesSupportsEinsteinConversationInsights	false
CapabilitiesSupportsKeyProvisioning	false
CapabilitiesSupportsNamedCredential	false
CapabilitiesSupportsPartnerContactCenterList	false
CapabilitiesSupportsPartnerPhoneNumbers	false
CapabilitiesSupportsPartnerTransferDestinations	false
CapabilitiesSupportsQueueManagement	true
CapabilitiesSupportsUniversalCallRecordingAccess	false
CapabilitiesSupportsUserSyncing	false
ConnectorUrl	https://my.connector.vendor.com/login
CustomLoginUrl	
DeveloperName	TestBYOTOrg
IntegrationClassId	01pxx0000004UQuAAM
Language	en_US
MasterLabel	Test BYOT Org
ModuleNamespace	
NamedCredentialId	[redacted]
TelephonySettingsComponent	
VendorType	ServiceCloudVoicePartner

You must follow all these instructions. Otherwise, the new feature isn't available and Salesforce falls back to no syncing behavior.

In addition to these instructions, customers still have to implement the Apex class using the sample code in the next section. Customers must specify the `IntegrationClassId` field in the `ConversationVendorInfo` record shown in the screenshot. If this value isn't set, an error message appears when trying to set up queue mapping.

Implement the Apex Class

Implement the Apex class using the sample code. Specify the class ID in the `IntegrationClassId` field of the `ConversationVendorInfo` record. If this value isn't set, an error message appears when trying to set up a queue mapping.

This code block contains a sample implementation of all interface methods. To interact with your queues, update the "TO DO" comments with your own implementation.

```
/**
* Sample code that implements QueueManager, QueueSetup, and GroupSetup in order to handle
* queue and group management requests.
*/
public class PartnerQueueManagementSampleClass implements service_cloud_voice.QueueManager,

                        service_cloud_voice.QueueSetup, service_cloud_voice.GroupSetup {


   /** ========== Sample code for methods defined in interface QueueManager ========== */



   /**
    * @description Returns whether the contact center supports user grouping.
    *              (Implementation for QueueManager.)
    * @param contactCenterInfo Info about the contact center.
    * @return QueueUserGroupingResponse Response containing whether the queue
    *                                    supports user grouping.
    */
   public service_cloud_voice.QueueUserGroupingResponse
supportsQueueUserGrouping(service_cloud_voice.ContactCenterInfo contactCenterInfo) {
       // Grab information from the request
       String contactCenterId = contactCenterInfo.getContactCenterId();


       // Returns whether contact center supports user grouping.
       // * @param boolean Indicates whether the method execution was successful.
       // * @param String Contains any error info that occurred during the
       // *               method execution.
       // * @param boolean Indicate if user grouping is supported.
       return new service_cloud_voice.QueueUserGroupingResponse(true, null, false);
   }

   /** ========== Sample code for methods defined in interface QueueSetup ========== */


   /**
    * @description Gets the list of queues. (Implementation for QueueSetup.)
    * @param queueListRequest Request containing information about retrieving queue list.

    * @return ListQueuesResponse Response containing the desired queues.
    */
   public service_cloud_voice.ListQueuesResponse
       listQueues(service_cloud_voice.ListQueuesRequest queueListRequest) {


       // TO DO: Call vendor's list queue API to retrieve the vendor queue list.
       //        The code below creates a dummy queue list.
       Map<String, String> queues = new Map<String, String>
         {'CustomerSupport' => 'External Customer Support',
           'ITSupport' => 'External IT Support',
           'FinancialSupport' => 'External Financial Support',
           'TechSupport' => 'External Tech Support'};

       // Returns the list of queues (for a successful response).
       // * @param boolean Indicates whether the method execution was successful.
       // * @param Map<String, String> Map of queues with queueKey <=> queue label pairs.

       // * @param String Contains any customer error message that occurred during the
       // *               method execution.
       return new service_cloud_voice.ListQueuesResponse(true, queues, null);


       // ERROR HANDLING
       // If an error occurs, you can return the error by passing false
       // in the response with an error message.
       //
       // Error Example (false response):
       //     return new service_cloud_voice.ListQueuesResponse(false,
       //            null, '<customized error message on expected error>');
   }
   /**
    * @description Creates a queue. (Implementation for QueueSetup.)
    * @param createQueueRequest Request containing information about creating a new queue.

    * @return CreateQueueResponse Response containing the new queue (or an error).
    */
   public service_cloud_voice.CreateQueueResponse
createQueue(service_cloud_voice.CreateQueueRequest createQueueRequest) {


       // Grab information from the request
       String contactCenterId =
createQueueRequest.getContactCenterInfo().getContactCenterId();
       String queueName = createQueueRequest.getQueueName();

       // TO DO: Call vendor's create queue API to create a new vendor queue and return
its id.
       //         The code below creates a dummy new queue ID.
       String queueId = 'TechSupport';

       // Returns the new queue ID (for a successful response).
       // * @param boolean Indicates whether the method execution was successful.
       // * @param String ID of the new queue.
       // * @param String Contains any error that occurred during the method execution.
       return new service_cloud_voice.CreateQueueResponse(true, queueId, null);


       // ERROR HANDLING
       // If an error occurs, you can return the error by passing false
       // in the response with an error message.
       //
       // Error Example (false response):
       //     return new service_cloud_voice.CreateQueueRequest(false,
       //             null, '<customized error message on expected error>');
   }


   /**
    * @description Removes an existing queue. (Implementation for QueueSetup.)
    * @param removeQueueRequest Request containing information about queue removal.
    * @return RemoveQueueResponse Response containing the queue removal information.
    */
   public service_cloud_voice.RemoveQueueResponse
removeQueue(service_cloud_voice.RemoveQueueRequest removeQueueRequest) {


       // Grab information from the request
       String contactCenterId =
removeQueueRequest.getContactCenterInfo().getContactCenterId();
       String queueId = removeQueueRequest.getQueueId();

       // TO DO: Call vendor's remove queue API to remove the existing vendor queue.
       // Returns the status of the removed queue (for a successful response).
       // * @param boolean Indicates whether the method execution was successful.
       // * @param String Contains any error info that occurred during the
       // *               method execution.
       return new service_cloud_voice.RemoveQueueResponse(true, null);


       // ERROR HANDLING
       // If an error occurs, you can return the error by passing false
       // in the response with an error message.
       //
       // Error Example (false response):
       //     return new service_cloud_voice.RemoveQueueResponse(false,
       //            null, '<customized error message on expected error>');
   }


   /**
    * @description Associates users and groups with a queue.
    * @param associateUsersAndGroupsWithQueueRequest Request containing
    *        information about the users and groups.
    * @return SyncUsersAndGroupsWithQueueResponse Response containing the result.
    */
   public service_cloud_voice.SyncUsersAndGroupsWithQueueResponse
associateUsersAndGroupsWithQueue(service_cloud_voice.SyncUsersAndGroupsWithQueueRequest
associateUsersAndGroupsWithQueueRequest) {


       // Grab information from the request
       String contactCenterId =

associateUsersAndGroupsWithQueueRequest.getContactCenterInfo().getContactCenterId();
       String queueId =
           associateUsersAndGroupsWithQueueRequest.getQueueId();
       List<service_cloud_voice.UserInfo> userInfoList =
           associateUsersAndGroupsWithQueueRequest.getUserInfoList();
       List<service_cloud_voice.GroupInfo> groupInfoList =
           associateUsersAndGroupsWithQueueRequest.getGroupInfoList();

      // TO DO: Call vendor's sync users and groups with queue API to add users and groups
 to
       //        the existing vendor queue.

       // Returns the status of the association (for a successful response).
       // * @param boolean Indicates whether the method execution was successful.
       // * @param String Contains any error info that occurred during the
       // *               method execution.
       return new service_cloud_voice.SyncUsersAndGroupsWithQueueResponse(true, null);


       // ERROR HANDLING
       // If an error occurs, you can return the error by passing false
       // in the response with an error message.
       //
    // Error Example (false response):
    //     return new service_cloud_voice.SyncUsersAndGroupsWithQueueResponse(false,
    //            null, '<customized error message on expected error>');


}

/** ========== Sample code for methods defined in interface GroupSetup ========== */


/**
 * @description Gets the list of groups. (Implementation for GroupSetup.)
 * @param groupListRequest Request containing information about retrieving group list.

 * @return ListGroupsResponse Response containing the desired groups.
 */
public service_cloud_voice.ListGroupsResponse
    listGroups(service_cloud_voice.ListGroupsRequest groupListRequest) {


    // TO DO: Call vendor's list group API to retrieve the vendor group list.
    //        The code below creates a dummy group list.
    Map<String, String> groups = new Map<String, String>
      {'CustomerSupportGroup' => 'External Customer Support Group',
        'ITSupportGroup' => 'External IT Support Group',
        'FinancialSupportGroup' => 'External Financial Support Group',
        'TechSupportGroup' => 'External Tech Support Group'};

    // Returns the list of groups (for a successful response).
    // * @param boolean Indicates whether the method execution was successful.
    // * @param Map<String, String> Map of groups with groupKey <=> group Label pairs
    // * @param String Contains any customer error message that occurred during the
    // *               method execution.
    return new service_cloud_voice.ListGroupsResponse(true, groups, null);


    // ERROR HANDLING
    // If an error occurs, you can return the error by passing false
    // in the response with an error message.
    //
    // Error Example (false response):
    //     return new service_cloud_voice.ListGroupsResponse(false,
    //            null, '<customized error message on expected error>');
    //
}


/**
 * @description Creates a group. (Implementation for GroupSetup.)
 * @param createGroupRequest Request containing information about creating a new group.

 * @return CreateGroupResponse Response containing the new group (or an error).
 */
public service_cloud_voice.CreateGroupResponse
```

```
createGroup(service_cloud_voice.CreateGroupRequest createGroupRequest) {


       // Grab information from the request
       String contactCenterId =
createGroupRequest.getContactCenterInfo().getContactCenterId();
       String groupName = createGroupRequest.getGroupName();

       // TO DO: Call vendor's create group API to create a new vendor group and return
its id.
       //        The code below returns a dummy new group ID.
       String groupId = 'TechSupportGroup';

       // Returns the new group ID (for a successful response).
       // * @param boolean Indicates whether the method execution was successful.
       // * @param String ID of the new group.
       // * @param String Contains any customer error message that occurred during the
method execution.
       return new service_cloud_voice.CreateGroupResponse(true, groupId, null);


       // ERROR HANDLING
       // If an error occurs, you can return the error by passing false
       // in the response with an error message.
       //
       // Error Example (false response):
       //     return new service_cloud_voice.CreateGroupResponse(false,
       //            null, '<customized error message on expected error>');
       //
   }


   /**
    * @description Associates users with a group.
    * @param associateUsersWithGroup Request containing information about the users.
    * @return SyncUsersWithGroupResponse Response containing the result.
    */
   public service_cloud_voice.SyncUsersWithGroupResponse
associateUsersWithGroup(service_cloud_voice.SyncUsersWithGroupRequest
associateUsersWithGroupRequest) {


       // Grab information from the request
       String contactCenterId =
           associateUsersWithGroupRequest.getContactCenterInfo().getContactCenterId();
       String groupId =
           associateUsersWithGroupRequest.getGroupId();
       List<service_cloud_voice.UserInfo> addedUserInfoList =
           associateUsersWithGroupRequest.getAddedUserInfoList();
       List<service_cloud_voice.UserInfo> removedUserInfoList =
           associateUsersWithGroupRequest.getRemovedUserInfoList();

       // TO DO: Call vendor's sync users with group API to add users to the
       //        existing vendor group.
```

```
        // Returns the status of the association (for a successful response).
        // * @param boolean Indicates whether the method execution was successful.
        // * @param String Contains any customer error message that occurred during the
        // *               method execution.
        return new service_cloud_voice.SyncUsersWithGroupResponse(true, null);


        // ERROR HANDLING
        // If an error occurs, you can return the error by passing false
        // in the response with an error message.
        //
        // Error Example (false response):
        //     return new service_cloud_voice.SyncUsersWithGroupResponse(false,
        //            null, '<customized error message on expected error>');
        //
    }
}
```

Change Status While on a Call

With the pending status change feature, reps can change their Omni-Channel status while on a call.

To use this feature, vendors should set the value of `hasPendingStatusChange` in `CapabilitiesResult` to `true`. After Salesforce receives a value of `true` from the `getCapabilities()` method, the Omni-Channel status change button is enabled for reps during a call.

Then, when `setAgentStatus` requests are made, Salesforce supports an additional parameter, `enqueueNextState`, and the vendor can implement the enqueue status change feature to support this behavior.

Tip: As of Winter '24 and later, reps can also change their Omni-Channel status after AgentWork is assigned but before they accept or decline the call. Partners should listen for and handle the SET_AGENT_STATUS event during this period between CALL_STARTED and CALL_CONNECTED.

SEE ALSO:

Service Cloud Connector API Reference

Transcribe Calls in Real Time

To see transcriptions in real time, add the Enhanced Conversation component to the Voice Call record page through the Lightning App Builder. You can add this component for orgs that use Service Cloud Voice with Amazon Connect. For orgs that use Service Cloud Voice with Partner Telephony, add this component only if your telephony provider supports transcription.

You can create and send transcripts for one voice call at a time or in bulk. To send transcripts for one voice call at a time, use the Create Transcript API. To send transcripts in bulk, use the Create Transcripts in Bulk API. You can also use the Connect REST API to upload or update transcripts.

The Create Transcript API and Create Transcripts in Bulk API use JWT authorization to communicate with Salesforce, so make sure to send the transcribed messages with a valid JWT token.

You can also use the Connect REST API to upload or update transcripts.

Review the following information before you configure real-time transcription:

- Review the transcript-related limitations specified in the Service Cloud Voice Limits and Limitations page.
- When specifying the `vendorCallKey` for a transferred call, use the `vendorCallKey` of the `parentVoiceCall`. That is, use the `vendorCallKey` of the voice call (`VoiceCall`) record for the call with the original rep.
- In consult call use cases where a consult call hasn't merged with a multiparty call, use the `vendorCallKey` of the voice call record created for the consult call. To transcribe a consult call after it has merged with a multiparty call, use the `vendorCallKey` of the first voice call record created for the multiparty call.
- During a conversation that involves a transfer, all calls in the conversation get transcripts from the entire call. After the call ends, the transcript disappears from the conversation body component for a brief period while the post-call association takes place.
- The post-call association process is triggered by the conversation closed event. This process ensures that the transcript for each call segment only includes the conversation during that time period. This process can take a few minutes.

SEE ALSO:

Service Cloud Voice Implementation Guide: Create a Transcript

Service Cloud Voice Implementation Guide: Create Transcripts in Bulk

Service Cloud Voice Implementation Guide: Upload or Update Transcripts with Connect REST API

Service Cloud Voice Implementation Guide: Increase Performance of the ContactLensProcessor Lambda Function

Salesforce Help: Service Cloud Voice Limits and Limitations

Send Real-Time Signals

Voice resiliency ensures that calls can still go through when the number of conversations is over limit or when the background service is affected.

Add support for sending real-time signals so that Salesforce can ingest the signals from your partner telephony system.

Perform the steps in this document to support sending real-time signals, giving administrators the ability to set up intelligent signals when they configure Conversation Intelligence rules. In the following steps, you will add support for sending real-time signals, implement the IntelligenceServiceProvider Apex interface and related classes, and then create and deploy the managed package for distribution to customers.

After the customer installs the managed package in their org, they can set the intelligence signal source and signal types when they create a Conversation Intelligence rule.

To add support for sending real-time signals, follow these instructions.

1. When you import the ConversationVendorInfo record, set the CapabilitiesSupportsIntelligence field to true to enable the feature.
2. Using the following code blocks as guides, implement the `service_cloud_voice.IntelligenceServiceProvider` Apex interface and related classes.

   IntelligenceServiceProvider Apex Interface. Implement this interface to add support for sending real-time signals.

```
global interface IntelligenceServiceProvider {
  /**
   * This method is to get a map of supported intelligence services and corresponding
signal types
   * @return IntelligenceServiceResponse
   */
  IntelligenceServiceResponse
getSupportedIntelligenceServicesAndSignalTypes(IntelligenceServiceRequest
intelligenceServiceRequest);
}
```

IntelligenceServiceRequest Apex Class. The following class definition represents the shape of the request payload that is needed for the `IntelligenceServiceResponse getSupportedIntelligenceServicesAndSignalTypes` method.

```
global with sharing class IntelligenceServiceRequest {
  private ContactCenterInfo contactCenterInfo;
  /**
   * Constructor for creating IntelligenceServiceRequest
   * @param contactCenterInfo contact center info
   */
  global IntelligenceServiceRequest(ContactCenterInfo contactCenterInfo) {
    this.contactCenterInfo = contactCenterInfo;
  }
  global ContactCenterInfo getContactCenterInfo() {
    return contactCenterInfo;
  }
}
```

IntelligenceServiceResponse Apex Class. This class definition represents the shape of the response that is returned from the `IntelligenceServiceResponse getSupportedIntelligenceServicesAndSignalTypes` method.

```
global with sharing class IntelligenceServiceResponse extends PartnerResponse {
  private List<IntelligenceServiceAndSignalsInfo> intelligenceServiceAndSignalsInfos;
  /**
   * Constructor for creating IntelligenceServiceResponse
   * @param intelligenceServiceInfos Map of supported intelligence services and
corresponding signal types
   */
  global IntelligenceServiceResponse(boolean success, String errorMessage,
List<IntelligenceServiceAndSignalsInfo> intelligenceServiceAndSignalsInfos) {
  super(success, errorMessage);
  this.intelligenceServiceAndSignalsInfos = intelligenceServiceAndSignalsInfos;
  }
  /**
   * @return supported intelligence services and corresponding signal types
   */
  global List<IntelligenceServiceAndSignalsInfo> getIntelligenceServiceAndSignalsInfos()
 {
    return this.intelligenceServiceAndSignalsInfos;
  }
}
```

IntelligenceServiceAndSignalsInfo Apex Class. This class definition represents an individual instance of an intelligence service along with a list of the supported signal types. The `IntelligenceServiceResponse` class contains this class.

```
global with sharing class IntelligenceServiceAndSignalsInfo {
  private String service;
  private String masterLabel;
  private Set<IntelligenceSignalType> signalTypes;
  /**
   * Constructor for creating IntelligenceServiceInfo.
   * @param List of Services and Supported SignalTypes for the Service.
   */
  global IntelligenceServiceAndSignalsInfo(String service, String masterLabel,
Set<IntelligenceSignalType> signalTypes) {
    this.service = service;
    this.masterLabel = masterLabel;
    this.signalTypes = signalTypes;
  }
  global String getService() {
    return this.service;
  }
  global String getMasterLabel() {
    return this.masterLabel;
  }
  global Set<IntelligenceSignalType> getSignalTypes() {
    return this.signalTypes;
  }
}
```

IntelligenceSignalType Apex Class. This enum represents the list of supported intelligence signal types. This class is used in the `IntelligenceServiceAndSignalsInfo` Apex class.

```
global enum IntelligenceSignalType {
  Category,
  Sentiment
}
```

Sample Implementation. Use the following sample to add your implementation to the `IntelligenceServiceProvider` class:

```
/**
* Adds support for sending real-time signals,
* <INTELLIGENCE SERVICE> represents the intelligence service that the partner is using.
 This name is unique to each partner. . For example, if the partner telephony vendor is
 NICE, set <INTELLIGENCE SERVICE> to CXoneAgentAssistService. If the partner telephony
 is another vendor, reach out to your PM to find the value for <INTELLIGENCE SERVICE>.
* <INTELLIGENCE SERVICE NAME> is the label name describing the intelligence service that
 the partner uses. For example,  if the intelligence service is CXone, set <INTELLIGENCE
 SERVICE NAME> to CXone Agent Assist Service.
*/
global class intelligenceService implements
service_cloud_voice.IntelligenceServiceProvider {
  global service_cloud_voice.IntelligenceServiceResponse
getSupportedIntelligenceServicesAndSignalTypes(service_cloud_voice.IntelligenceServiceRequest
 intelligenceServiceRequest) {
    List<service_cloud_voice.IntelligenceServiceAndSignalsInfo>
intelligenceServiceAndSignalsInfos = new
List<service_cloud_voice.IntelligenceServiceAndSignalsInfo>();

    Set<service_cloud_voice.IntelligenceSignalType> signalTypes = new
Set<service_cloud_voice.IntelligenceSignalType>();
    signalTypes.add(service_cloud_voice.IntelligenceSignalType.Sentiment);

    intelligenceServiceAndSignalsInfos.add(new
service_cloud_voice.IntelligenceServiceAndSignalsInfo(‘<INTELLIGENCE SERVICE>’,
‘<INTELLIGENCE SERVICE NAME>’, signalTypes));

    return new service_cloud_voice.IntelligenceServiceResponse(true, null,
intelligenceServiceAndSignalsInfos);
   }
}
```

3. Prepare the managed package.
   - If you've never deployed a managed package for Service Cloud Voice, deploy the managed package now.
   - If you've deployed a managed package for Service Cloud Voice before, update the managed packages with your implementation of the IntelligenceServiceProvider class.

   After you prepare the managed package, copy the URL and send it to your customer, who will use it to install the managed package in their org.

Supervisor Listen In or Barge In with Service Cloud Voice for Partner Telephony

When a rep belongs to a supervised group, a supervisor can monitor their active calls using the Supervisor Panel.

When a supervisor clicks **Monitor** on an active call, in addition to viewing the conversation entries (and real-time transcription), the supervisor can join and listen in, muted. If supported, the supervisor can also barge in (unmute), or disconnect.

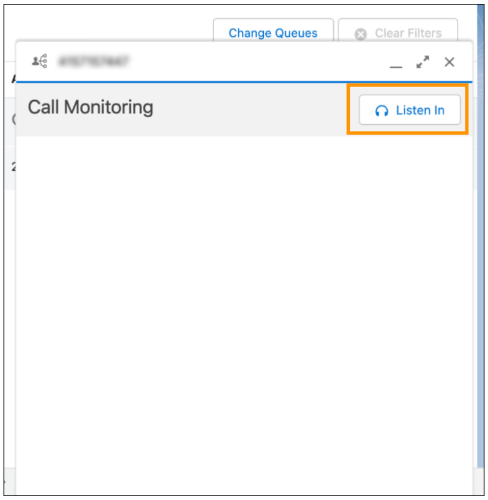
For listening in to work, the supervisor can load the vendor's connector using the Omni-Channel widget. The supervisor must belong to the contact center that they're supervising.

In addition to these requirements, the supervisor must be assigned the SCV supervisor user permission and also have the value of the `CapabilitiesResult` field `hasSupervisorListenIn` set to `true`. In order for the supervisor to join a call unmuted, `hasSupervisorBargeIn` must also be set to `true`

In order for a supervisor to listen in or barge in:

1. In Permission Sets, add the App Permission "Contact Center Supervisor" to the user in addition to "Contact Center Agent (Partner Telephony)".
2. When implementing `getCapabilities()`, have `hasSupervisorListenIn` set to `true` in the returned `CapabilitiesResult`. If you want to allow the supervisor to barge in, unmuted, set `hasSupervisorBargeIn` to `true`.
3. Have the Omni-Channel widget available for this user.
4. Add the supervisor to the same contact center that they're supervising.
5. Create an Omni-Channel Supervisor configuration.
6. Add Omni-Channel Supervisor to the Salesforce console.
7. Add Omni-Channel Supervisor to a Lightning console app.
8. Set up a supervisor queue.

After the setup is complete, a supervisor can see the **Listen In** button enabled when monitoring a Voice Call.



Supervisor Listen In Details

When a supervisor clicks the **Listen In** button, the call information is delegated to the vendor's connector `superviseCall(supervisedCallInfo)` method. The `superviseCall(supervisedCallInfo)` method creates a call leg between the supervisor and the parent call, updates the list of active calls, and returns a promise of type `SupervisorCallResult`.

If the supervisor is on a hard phone, after the returned promise is resolved successfully, the supervisor call leg enters a Dialing state. The hard phone is expected to publish the `SUPERVISOR_CALL_STARTED` event after the Supervisor picks up the headset.

If the Supervisor is on a softphone, after the returned promise is resolved successfully, the supervisor call leg enters the Connected state. No extra events should be fired.

When the supervisor leaves the call using the softphone, the call information is delegated to the vendor's connector `supervisorDisconnect(call)` method. This method must create a call leg between the supervisor and the parent call and return a promise. The vendor implementation should destroy the supervisor leg and update the list of active calls.

When the supervisor leaves the call using the hard phone, the vendor implementation should destroy the supervisor leg and update the list of active calls, and fire the `SUPERVISOR_HANGUP` event.

For example:

```
// This is used when using deskphone as call controls (not Omni-Channel)

// Called by deskphone when initiating the SV call - the call starts ringing
superviseCallonHardphone(call){
   return await sdk.superviseCall(call);
}

// Called by deskphone after SV picks up on deskphone
connectSupervisedCallonHardphone(call){
  try {
    const result = await sdk.connectSupervisedCall(call);
    publishEvent({ eventType: constants.VOICE_EVENT_TYPE.SUPERVISOR_CALL_CONNECTED, new
SuperviseCallResult(call) });
  } catch (e) {
    publishError(constants.VOICE_EVENT_TYPE.SUPERVISOR_CALL_CONNECTED, e);
  }
}
```

Supervisor Barge In Details

When a supervisor barges in, we use the same VoiceCall record. The participant type is created automatically and is reused for transcription.

```
POST /voiceCalls/${vendorCallKey}/messages, {
   messageId,
   content,
   senderType: "SUPERVISOR",
   startTime,
   endTime,
   participantId: "supervisorId"
}, headers
```

When a supervisor barges in, you can reuse the toggle `callInfo.recordEnabled` and the capability `CapabilitiesResult.hasRecord` to control whether the supervisor can toggle a recording. Specifically, a recording toggle is allowed if all these conditions are met:

1. `CapabilitiesResult.hasRecord` is true.
2. `callInfo.recordEnabled` is true.
3. `CapabilitiesResult.hasSupervisorListenIn` and `CapabilitiesResult.hasSupervisorBargeIn` are both true.
4. The call is connected.

When clicked, we call the same `pauseRecording()` and `resumeRecording()` functions.

SEE ALSO:

Service Cloud Connector API Reference

Send Additional Call Information

You can disable certain call handling options such as end call, dial pad, and phone book in the softphone.

Use the CALL_UPDATED event to send additional information during a call to disable certain call handling options such as end call, dial pad, and phone book in the softphone. This event uses the CallResult class object, so you can use the CallInfo object to update the softphone controls. This event can be published during the call.

When implementing the getVoiceCapabilities(), set the value of the required field of the VoiceCapabilitiesResult to disable the corresponding option in the softphone. For example, to disable the dial pad, when implementing getVoiceCapabilities(), set the value of the isDialPadDisabled field of VoiceCapabilitiesResult to true.

CHAPTER 11 Post-Call Actions

In this chapter ...

This section provides guidelines about how to fine-tune post-call actions.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

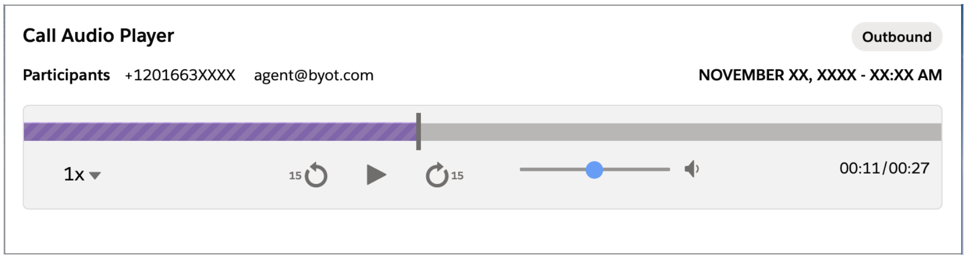
Call Recordings

Each voice call supports one call recording. If a participant is added to the call, each of the component voice calls can be updated to list the recording URL and duration details.

During the Post-Call CTR Sync on page 91, you can call the Update Voice Call API with the following fields to add call recording information to a voice call.

- **recordingLocation:** A complete call recording URL. This URL belongs to your domain, and can be accessed only if the rep is logged into the telephony system through SSO or a login page. Otherwise, the rep sees a message indicating that they don't have access or need to log into the phone system. This URL can be a public URL or a URL only accessible through a rep SSO or login page.
- **totalHoldDuration:** The total number of seconds the call was on hold. If an audio recording doesn't include hold time, this value is zero.
- **agentInteractionDuration:** The total number of seconds the call wasn't on hold.

Note: The call recording duration shown in the Call Audio Player is the sum of the agentInteractionDurationand and totalHoldDuration values. If this sum doesn't equal the recording's actual duration, the recording may appear to end before or after the boundaries of the slider.


Call Audio Player
Outbound
Participants +1201663XXXX agent@byot.com
NOVEMBER XX, XXXX - XX:XX AM
1x
15
15
00:11/00:27

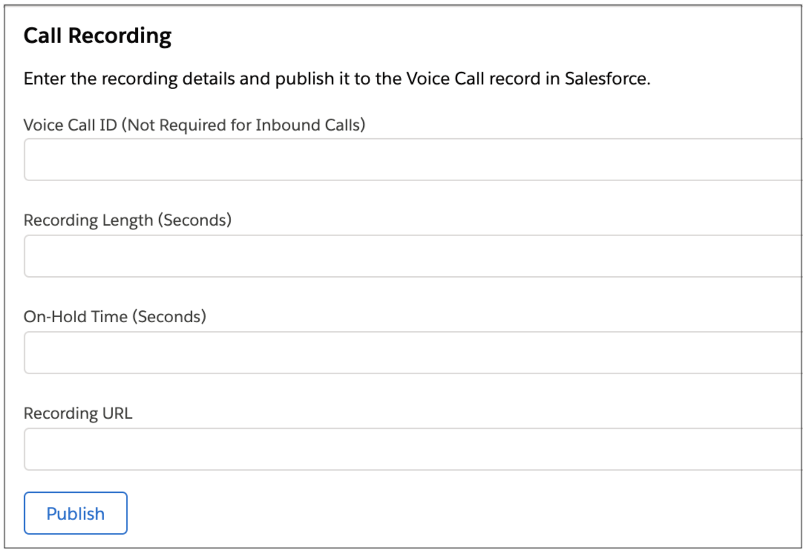
The callDurationInSeconds field on voice calls is determined by calculating the difference between the startTime and endTime fields, and is unrelated to call recording duration.

Note: The date in the top-right corner of the media player is the call's start time, and it matches the start time shown in the Voice Call details. If the date doesn't display properly on an inbound call, verify that:

- The start time is sent correctly when calling the Create Voice Call API
- The start time in the Voice Call details is correct

Trusted URL: To allow recordings to be loaded, add your domain as a Trusted URL in Salesforce Setup. To learn more, see Manage Trusted URLs.

Example and testing: See Use the Demo Connector on page 32 for a demonstration of calling Update Voice Call API with a recording URL and duration details. The recording link is capped at 1,000 characters and should not exceed this length. The Call Recording section in the Voice Call Simulator lets you experiment with the recording feature.



Call Recording

Enter the recording details and publish it to the Voice Call record in Salesforce.

Voice Call ID (Not Required for Inbound Calls)

Recording Length (Seconds)

On-Hold Time (Seconds)

Recording URL

getSignedRecordingUrl

In order to provide recording URLs based on the vendorCallKey, Salesforce Voice Call ID, or recordingUrl inserted via Update Voice Call API, vendors can implement Service Cloud Connector API getSignedRecordingUrl in their connectors. This API accepts vendorCallKey, callId, recordingUrl as parameters and returns a SignedRecordingUrlResult object.

The vendor specifies *hasSignedRecordingUrl* as true. When the recording component loads, Salesforce issues a request to the Service Cloud Connector API getSignedRecordingUrl to get the recording URL for the Voice Call. Using this method, the vendor can provide updated recording URLs when requested by the Voice Call Recording component.

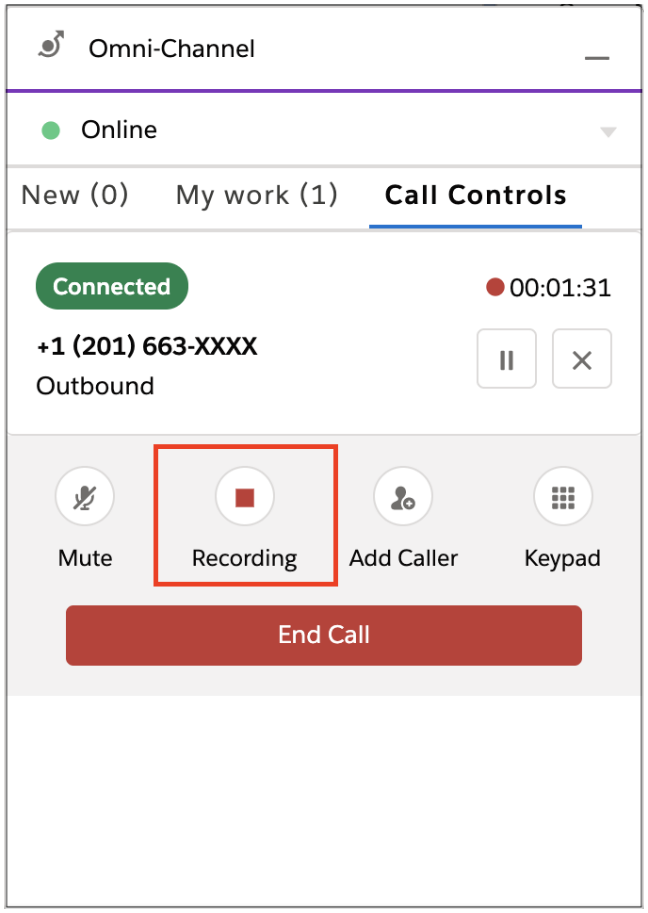
```
// Create a signed recording url capability
function getCapabilities() {
   return new CapabilitiesResult({
        hasSignedRecordingUrl: true
   });
}

function getSignedRecordingUrl(recordingUrl, vendorCallKey, callId) {
   // Implementation goes in here
}
```

Pause and Resume Recording

Reps can pause and resume recording while on a call. This feature can be used when call participants want to share confidential details and don't want them to be recorded.

Implement these Service Cloud Connector APIs to allow users to pause and resume recording:

- pauseRecording(call)
- resumeRecording(call)



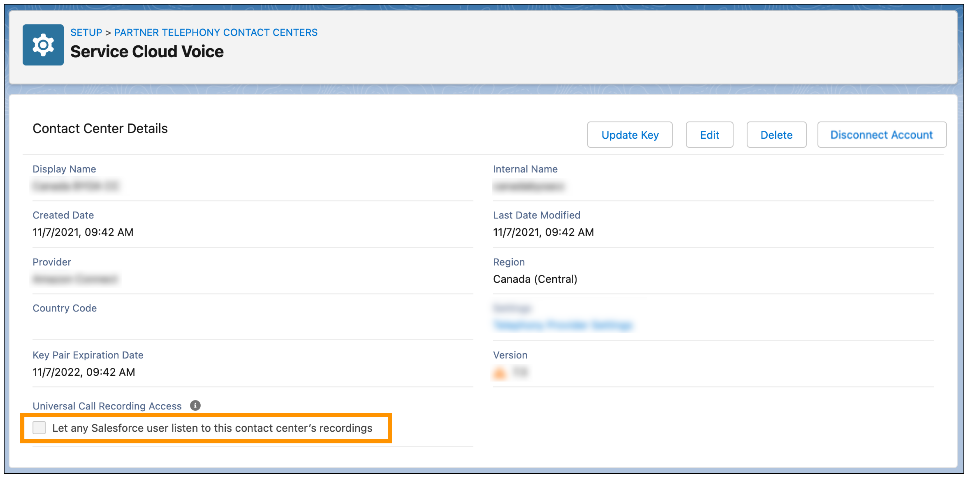
Universal Call Recording Playback

Contact centers can be configured so that you can share voice calls (and the associated voice call recordings) with other users in your contact center. This feature can be enabled with the **Universal Call Recording Access** checkbox on the contact center record details page.

To view this configuration setting, the following requirements must be met:

1. Partners must set the `supportsUniversalCallRecordingAccess` capability to true in their ConversationVendorInfo object.
2. The contact center should be connected to the partner telephony system.
3. The Apex integration class must implement the `RecordingMediaProvider` interface. This interface includes the `getSignedUrls` method, which is called to fetch the signed recording URL from the partner telephony system for loading the voice call recording playback.

After these requirements are met, check **Let any Salesforce user listen to this contact center's recordings**.



SETUP > PARTNER TELEPHONY CONTACT CENTERS

Service Cloud Voice

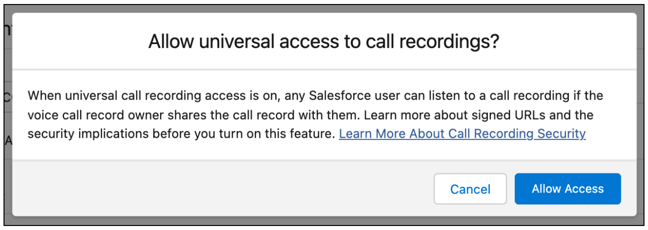
Contact Center Details

Display Name [redacted]	Internal Name [redacted]
Created Date 11/7/2021, 09:42 AM	Last Date Modified 11/7/2021, 09:42 AM
Provider [redacted]	Region Canada (Central)
Country Code	[redacted]
Key Pair Expiration Date 11/7/2022, 09:42 AM	Version [redacted]

Universal Call Recording Access

□ Let any Salesforce user listen to this contact center’s recordings

When a warning dialog appears, click **Allow**.



Allow universal access to call recordings?

When universal call recording access is on, any Salesforce user can listen to a call recording if the voice call record owner shares the call record with them. Learn more about signed URLs and the security implications before you turn on this feature. Learn More About Call Recording Security

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference

Service Cloud Connector API Reference: Get Started with the Service Cloud Connector API

Post-Call CTR Sync with the Update VoiceCall API

This API updates a VoiceCall record after the call has ended. Use this API to update call-related parameters that are unavailable during the VoiceCall creation stage, such as callDuration and numberOfHold.

The Update VoiceCall API is an asynchronous operation. You can’t query for the status of the API call. To learn more, see Update VoiceCall.

For production orgs you can use the Service Cloud Voice REST API: `https://{org_domain_name}.my.salesforce-scrt.com/telephony/v1`.

Use the org domain name as defined in the first step of Set Up SSO Connected App and Salesforce with Identity Provider on page 40.

This API uses JWT authorization to communicate with Salesforce.

SEE ALSO:

Voice Implementation Guide: Update VoiceCall

After Conversation Work

After Conversation Work (ACW) gives reps a set amount of time after a customer conversation to wrap up their work before they start a new conversation.

In order for reps to configure After Conversation Work, follow these steps to support the feature:

1. Fire an AFTER_CALL_WORK_STARTED event (using the `publishEvent()` method) with the callId to trigger after conversation work for the rep.
   a. Fire this event regardless of whether the org has enabled After Conversation Work. In case After Conversation Work isn't enabled, this event is ignored.
   b. If there's a case (for example, a missed call) when the rep isn't put into wrap-up mode in the telephony system, the connector should not fire the AFTER_CALL_WORK_STARTED event.

```
publishEvent({ eventType: Constants.VOICE_EVENT_TYPE.AFTER_CALL_WORK_STARTED,
  payload: { callId: <uniqueCallId> }
});
```

2. Implement the `wrapUpCall()` method in the connector. The implementation of this method should remove the rep from wrap-up and put the rep back into the queue for receiving calls.
3. If you have Sales Engagement setup in your org and you'd like to advance a cadence after your rep wraps up a call, fire `WRAP_UP_ENDED` (using the `publishEvent()` method) event after you receive `wrapUpCall()`, with the following sample payload.

```
{
 callType: 'outbound' ,
 callId: <callId>,
 callStatus: 'ended'
}
```

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference: Get Started with the Service Cloud Connector API

Service Cloud Connector API Reference: wrapUpCall

Service Cloud Connector API Reference: publishEvent

Salesforce Help: Configure After Conversation Work Time

Release Notes: Give Agents Time for After-Conversation Work (Pilot)

Mean Opinion Score (MOS)

Support MOS using Service Cloud Voice for Partner Telephony.

The Mean Opinion Score (MOS) is a commonly used metric to measure the overall voice call quality. The MOS is a rating from 1 to 5, with 1 being the lowest score, and 5 being the highest. This rating is standardized by the International Telecommunications Union ITU-T.

The standard ITU-T G.107 defines the algorithm we use to calculate the G.711 MOS score, based on network performance metrics (for example, latency, jitter and packet loss).

In order to support MOS as a Service Cloud Voice partner:

1. Specify rep configuration the field `config.supportsMos` using the `setAgentConfig` method.
2. Publish the `UPDATE_AUDIO_STATS` event using the `publishEvent` method to report the audio stats during the call (between the call getting connected and ending). Stats are ignored before the call is connected and after you specify `isAudioStatsCompleted` as `true` in the `AudioStats` payload of the `publishEvent` method.

In order to get the MOS:

1. The audio stats are collected during the call. The MOS is calculated when the connector specifies `isAudioStatsCompleted` as `true` in the last `UPDATE_AUDIO_STATS` `publishEvent` payload. The value is stored in the Mean Opinion Score in the VoiceCall record.
2. The Salesforce admin must add the Mean Opinion Score in the VoiceCall layout to show the score in VoiceCall record UI.

SEE ALSO:

Service Cloud Connector API Reference

CHAPTER 12 Route Calls

In this chapter ...

This section provides guidelines related to routing calls.

❗ Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Omni-Channel Flow for Service Cloud Voice for Partner Telephony

Run an Omni-Channel flow using the Service Cloud Voice Telephony Integration REST API.

To learn more, see Execute OmniFlow in the Service Cloud Voice Implementation Guide.

Note: Executing an Omni-Channel Flow for incoming calls (by calling `/telephony/v1/voiceCalls/{CALL_ID}/omniFlow`) creates a PendingServiceRouting (PSR) record in Salesforce which stays in the queue until the voice call is routed to the rep and the rep accepts the AgentWork. If the voice call is never routed to the rep or the rep declines the AgentWork, the PSR record stays in Salesforce. In order to clean up a PSR record associated with an abandoned or declined voice call, call the Clear Routing API (`/telephony/v1/voiceCalls/{CALL_ID}/clearRouting`) to clean it up.

SEE ALSO:

Salesforce Help: Omni-Channel Flows

Service Cloud Voice Implementation Guide: Execute OmniFlow

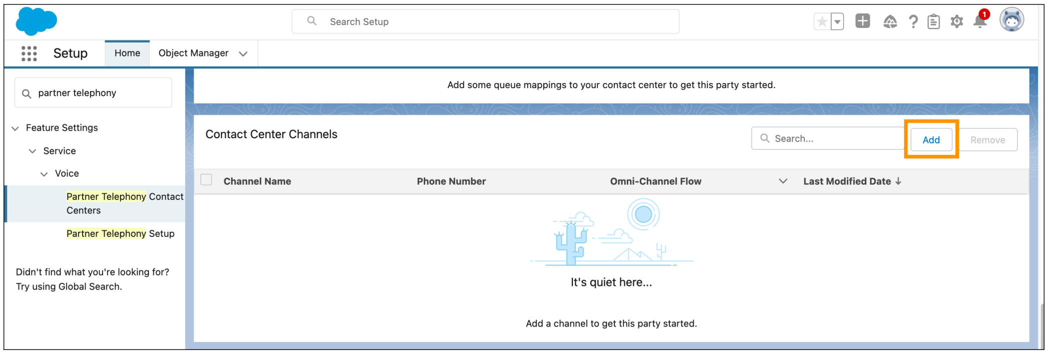
Service Cloud Voice Implementation Guide: Clear Routing

Salesforce Help: The Routing Lifecycle

Add Contact Center Channels to Enable Routing

To let customers configure call routing for voice calls and to determine when to create an End User record, create a phone channel. After creating a phone channel, you can set up a caller ID tool to create or reuse an End User record. You can also choose whether to associate the End User record with the number dialed.

Contact center channels can be added from the contact center details page. From Setup, enter `Partner Telephony` in the Quick Find box, then select **Partner Telephony Contact Centers**. Select your contact center and then scroll to the **Contact Center Channels** section. Click **Add**.



Add some queue mappings to your contact center to get this party started.

Contact Center Channels

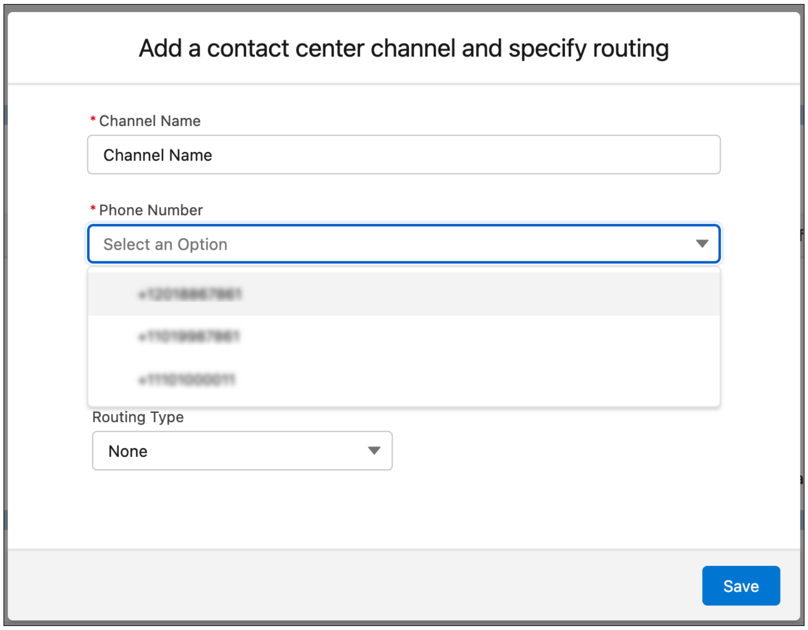
Channel Name	Phone Number	Omni-Channel Flow	Last Modified Date ↓

It's quiet here...

Add a channel to get this party started.

Didn't find what you're looking for? Try using Global Search.

Partners can automate the phone number selection by providing the phone number list through an Apex implementation. Otherwise, the phone input box displays a text box where the rep can manually enter the phone number.



Add a contact center channel and specify routing

* Channel Name

Channel Name

* Phone Number

Select an Option

[redacted]

[redacted]

[redacted]

Routing Type

None

Partners can turn on the capability `CapabilitiesSupportsPartnerPhoneNumbers` in ConversationVendorInfo and implement the `service_cloud_voice.PhoneNumberProvider` interface in their Apex class on page 125 as shown in the following code sample.

```
public service_cloud_voice.PhoneNumberResponse
listPhoneNumbers(service_cloud_voice.PhoneNumberRequest phoneNumberRequest) {
  List<service_cloud_voice.PhoneNumberInfo> phoneNumberInfos = new
List<service_cloud_voice.PhoneNumberInfo>();
  phoneNumberInfos.add(new service_cloud_voice.PhoneNumberInfo('+12018867861',  'US',
'TOLL_FREE',  '12018867861'));
  phoneNumberInfos.add(new service_cloud_voice.PhoneNumberInfo('+11019987861',  'US',
'TOLL_FREE',  '11019987861'));
  phoneNumberInfos.add(new service_cloud_voice.PhoneNumberInfo('+11101000011',  'US',
'TOLL_FREE',  '11101000011'));
  return  new service_cloud_voice.PhoneNumberResponse(true, null, phoneNumberInfos);
}
```

`PhoneNumberInfo` contains phoneNumber, countryCode, and an identifier.

Queue Mapping and Rep Mapping

This topic provides guidance on how to handle queue mapping and rep mapping.

Queue Mapping

Partners can provide a list of queues at runtime, which can be mapped by the admin to external queues in Salesforce using contact center UI. This mapping is used when a flow executes and returns a queue. In order to provide a list of queues for the UI, the partner package has to perform the following steps:

1. Set `partnerTransferDestinationsSupported` to `true` in the partner `ConversationVendorInfo` record.
2. Implement the Apex interface service_cloud_voice.TransferDestinationProvider on page 125.

Rep Mapping

Partners who support user syncing are able to support rep mapping in Salesforce. There's currently no UI to see the rep mapping, but the mapping entries are stored in `CallCenterRoutingMap` in Salesforce. This mapping is used when a flow executes and returns a rep. In order to support rep mapping, the partner package has to perform the following steps:

1. Set `userSyncingSupported` to `true` in the partner `ConversationVendorInfo` record.
2. Implement the Apex interface service_cloud_voice.UserSyncing on page 125, and return a unique user ID from the partner system in `UserSyncingResponse`.

SEE ALSO:

Set Up Service Cloud Voice for Partner Telephony in Your Org

Salesforce Help: Omni-Channel Flows

Enable the Voice Extension Page in Lightning App Builder

Configure this feature to let administrators add custom voice controls to their Omni-Channel softphones using the Voice Extension FlexiPage in the Lightning App Builder.

Administrators can create and customize Voice Extension pages, then enable them in a contact center so that any rep in the org can use the feature. For steps on how to enable and configure the Voice Extension Page, see the Enable the Voice Extension Page in Lightning App Builder page in the Service Cloud Voice Implementation Guide.

SEE ALSO:

Salesforce Help: Customize Call Controls and Voice Extensions

Understand Rep Statuses

See how `setAgentStatus()` is called from the Salesforce core to the connector when a rep performs an action in the Salesforce phone control panel.

This table explains how rep statuses are represented in Omni-Channel and in the connector.

Rep Action in Salesforce	Rep Status in Omni	SetAgentStatus sent to Connector	Connector Event/Callback	Rep Status in Vendor/Queue
Log in as offline	Offline	YES	SetAgentStatus	Offline/NotAvailableForRouting
Log in as online	Online	YES	SetAgentStatus	Online/AvailableForRouting
Make outbound call	Online	NO	dial()	NotAvailableForRouting
During outbound call	Online	NO	CALL_CONNECTED	NotAvailableForRouting
During After Conversation Work (ACW) after outbound call	Online	NO	HANG_UP/endCall()	NotAvailableForRouting

Rep Action in Salesforce	Rep Status in Omni	SetAgentStatus sent to Connector	Connector Event/Callback	Rep Status in Vendor/Queue
Close Voice Call tab during ACW after outbound call	Online	YES	SetAgentStatus	AvailableForRouting
Receive inbound call	Online	NO	CALL_STARTED	NotAvailableForRouting
Accept inbound call	Online	NO	acceptCall()	NotAvailableForRouting
Decline inbound call	Online	NO	declineCall()	AvailableForRouting
During inbound call	Online	NO	CALL_CONNECTED	NotAvailableForRouting
During ACW after inbound call	Online	NO	HANG_UP/endCall()	NotAvailableForRouting
Close Voice Call tab during ACW after inbound call	Online	YES	SetAgentStatus	AvailableForRouting

Note: The telephony system should be in an infinite wrap-up for all calls (including for After Conversation Work). This way, when a rep finishes a call, they aren't added to the queue until they're made available by logging in as online or by closing the Voice Call tab during an After Conversation Work period.

SEE ALSO:

Service Cloud Connector API Reference: Get Started with the Service Cloud Connector API

Service Cloud Connector API Reference: setAgentStatus

Two-Way Rep Status Syncing

Rep status (that is, rep presence) can be changed from the Omni-Channel widget, which sends status information to the partner connector. We added the ability to change the rep status from the connector, which passes status information back to Salesforce.

For example, a telephony partner system may have scheduled time breaks for the reps where they would want to change the rep status in Omni accordingly. If a rep is on a call, status change is ignored and doesn't have any impact. In order to support complete two-way syncing of the status between Omni and the telephony system, the telephony system should persist a table with mapping between the Salesforce status ID and the partner status.

When the connector loads, the `init()` API is called, and the argument `callCenterConfig` contains a JSON field called `userPresenceStatuses` that can be parsed into a map of `statusId: statusInfo`. For example:

```
{
    "0": {
        "statusName": "Offline",
        "hasChannels": "false",
        "isOffline": "true",
        "statusId": "0"
    },
    "0N5xx0000004CSO": {
        "statusName": "Available",
        "hasChannels": "true",
        "isOffline": "false",
        "statusId": "0N5xx0000004CSO"
    },
    "0N5xx0000004D56": {
        "statusName": "Away",
        "hasChannels": "false",
        "isOffline": "false",
        "statusId": "0N5xx0000004D56"
    },
    "0N5xx0000004D3U": {
        "statusName": "Busy",
        "hasChannels": "false",
        "isOffline": "false",
        "statusId": "0N5xx0000004D3U"
    }
}
```

The statusId 0 is reserved for the `Offline` status. Other statusIDs represent Salesforce Omni-Channel presence statuses that are available for the user.

The statusInfo fields are:

- `statusName`: The name of the status.
- `hasChannels`: `false` indicates that the status is busy or offline. `true` indicates that it's routable by a channel.
- `isOffline`: `true` indicates that it's an offline status (with statusId 0).
- `statusId`: Salesforce status ID

In order to invoke the status change from the connector, call `publishEvent()` with the event `SET_AGENT_STATUS` and the required status ID. For example:

```
Const statusId = "0N5xx0000004D3U";
publishEvent({ eventType: "SET_AGENT_STATUS", payload: new AgentStatusInfo({ statusId })
});
```

To change the status to offline:

```
Const offlineStatusId = "0";
publishEvent({ eventType: "SET_AGENT_STATUS", payload: new AgentStatusInfo({ statusId:
offlineStatusId })});
```

See `setAgentStatus()` for information on how to update the vendor rep status when the Omni-Channel status changes.

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference

Handling Missed Calls and Call Errors

A voice call typically ends when a rep hangs up. Voice calls can also end when an rep misses the call or when the call is in an error state. This topic shows you how missed voice calls, including inbound, outbound, transfer, and callback calls, and call errors are handled.

To end a call, `call.reason` must be set to `ended` or `error`, and `call.closeCallOnError` must be set to `true` or `false`.

Calls usually end when a rep clicks the **End Call** button, invoking the vendor's Connector API endCall() method. In such cases, configure the connector to raise a `HANGUP` event with `call.reason` set to `ended` to mark the voice call as completed.

However, calls can also end when a rep misses or declines the call, or when the call fails and is in an error state.

For missed, declined, or failed calls, configure the connector to:

- Raise a `HANGUP` event type with `call.reason` set to error in the Connector API publishEvent() method.
- Set `call.closeCallOnError` to false to keep the conversation open and leave the voice call in a "new" state. Don't set `call.closeCallOnError` to true unless you want to close the conversation and mark the voice call as completed.

For missed, declined, or failed calls, also set the following for inbound or callback calls:

- When a rep misses a call, set `call.agentStatus` to `MissedCallAgent`.
- When a rep declines a call, set `call.agentStatus` to `DeclinedByAgent`.
- When a call fails due to the rep, set `call.agentStatus` to `FailedConnectAgent`.
- When a call fails due to any reason that's unrelated to the rep, set `call.agentStatus` to `FailedConnectCustomer`.

To set the rep's status automatically when the call is declined, go to **Setup | Presence Configuration Settings | Update Status on Decline**, and choose a presence status for when the rep declines a work item.

To set the rep's status automatically when the call is missed or if there's any error that isn't due to call declined, go to **Setup | Presence Configuration Settings | Update Status on Push Time-Out** and choose a presence status.

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference: Get Started with the Service Cloud Connector API

Service Cloud Connector API Reference: publishEvent

Service Cloud Connector API Reference: CallResult

Service Cloud Connector API Reference: PhoneCall

External Routing

While Service Cloud Voice uses Omni-Channel routing capabilities, in a partner telephony integration Salesforce typically handles the creation of AgentWork records for voice calls based on call events sent by the partner.

For external routing of voice calls, the partner system identifies the rep which the call should be routed to. The partner sends a call started event for that particular rep. If needed, the partner reroutes the call to another rep.

Partners can also route calls by using an Omni-Channel flow and handling routing through their partner system. Flows can route directly to a rep or to a Salesforce Queue. For more information, see Omni-Channel Flow for Service Cloud Voice for Partner Telephony on page 95.

The standard Omni-Channel External Routing flow, where the partner explicitly creates the AgentWork record using the PendingServiceRouting (PSR), applies primarily to non-voice channels or specific advanced unified routing configurations. To implement external routing for non-voice channels, refer to Omni-Channel documentation. For detailed guidance on implementing External Routing for non-voice channels using Omni-Channel, see the External Routing for Omni-Channel section in the Omni-Channel Developer Guide.

To learn more about the PendingServiceRouting (PSR), see The Routing Lifecycle in Salesforce Help.

SEE ALSO:

Salesforce Help: The Routing Lifecycle

Unified Routing (Beta)

Unified routing lets Salesforce handle the routing of inbound and outbound voice calls, skills-based call routing, and call transfers to the reps. Configure unified routing so that the admins can enable the feature. You can choose whether to include this feature in the managed package.

Note: Unified routing is a pilot or beta service that is subject to the Beta Services Terms at Agreements - Salesforce.com or a written Unified Pilot Agreement if executed by Customer, and applicable terms in the Product Terms Directory. Use of this pilot or beta service is at the Customer's sole discretion.

To configure unified routing, set the `CapabilitiesSupportsUnifiedRouting` field of the ConversationVendorInfo object to true, and invoke the omni flow. The admins can then enable unified routing for the contact centers from the contact center details page. Once the supportsUnifiedRouting field is set to true, it can't be changed to false.

When unified routing is enabled, Salesforce handles the inbound and outbound voice call routing, skills-based call routing, and call transfers to the reps. When unified routing is enabled, place the calls in a holding or temporary queue.

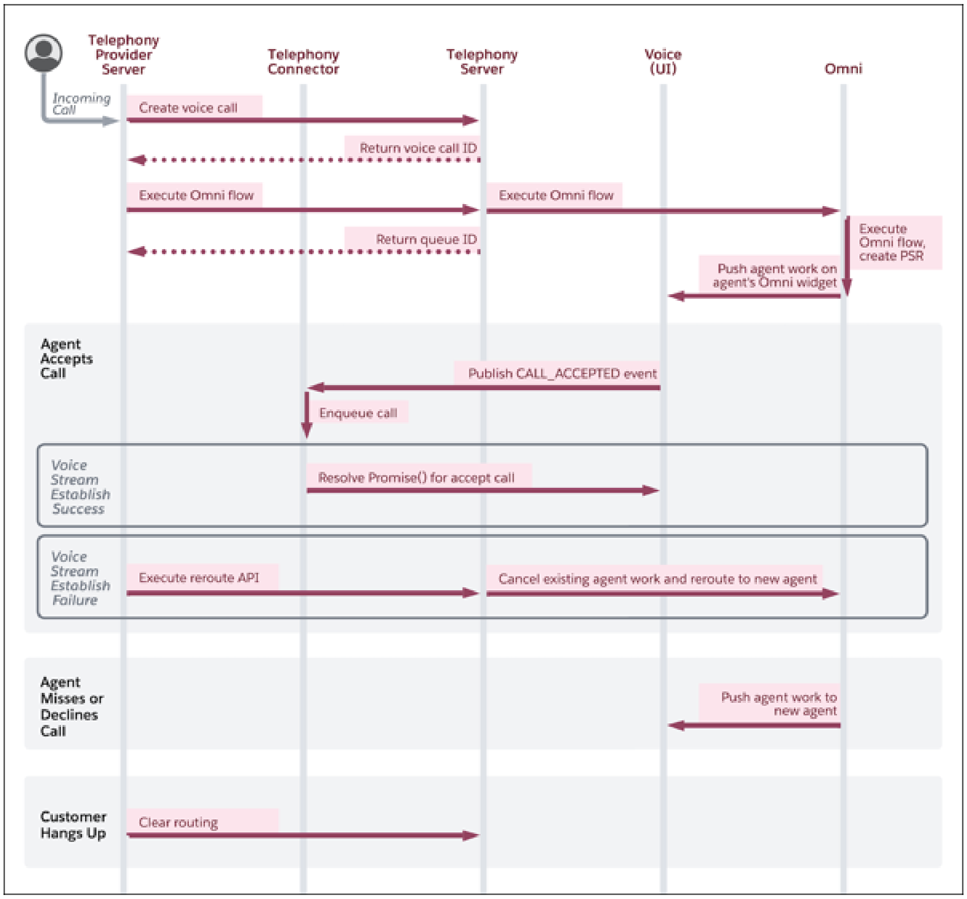
If you want to use the sample implementation in the Demo Connector, select the Unified Routing capability in the Routing Settings to enable unified routing. Also provide the omni flow dev name and the Salesforce fallback queue id in the Demo Connector.

Note: Unified routing doesn't support transfer by flow.

Unified Routing Flow

Only if you provide unified routing support in the managed package, Salesforce can handle call routing.

The high-level unified routing flow is:


Telephony Provider Server
Telephony Connector
Telephony Server
Voice (UI)
Omni
Incoming Call
Create voice call
Return voice call ID
Execute Omni flow
Execute Omni flow
Return queue ID
Execute Omni flow, create PSR
Push agent work on agent's Omni widget
Agent Accepts Call
Publish CALL_ACCEPTED event
Enqueue call
Voice Stream Establish Success
Resolve Promise() for accept call
Voice Stream Establish Failure
Execute reroute API
Cancel existing agent work and reroute to new agent
Agent Misses or Declines Call
Push agent work to new agent
Customer Hangs Up
Clear routing

- Receive incoming call: A customer initiates a call, which the telephony provider server receives.
- Create voice call: The telephony provider server sends a request to create a voice call to the Salesforce telephony service. The Salesforce telephony service processes the request and returns a Voice Call ID.
- Execute Omni-Channel flow: The telephony provider server executes the Omni-Channel Flow by invoking the Salesforce Telephony Service's Execute an Omni Flow API Call.
- Route Call and notify rep: Salesforce Omni-Channel executes the necessary flow to identify and route the call to the most appropriate rep. A PendingServiceRouting (PSR) record is created to track the routing process. Voice pushes the work to the rep's Omni-Channel widget in Salesforce, notifying the rep of an incoming call.
- Accept or decline the call.
  - Rep accepts the call: The rep's browser publishes a `CALL_ACCEPTED` event to the third-party vendor connector. The third-party vendor connector must assign the audio stream to the given rep.
    - Audio stream establishment: The vendor connector attempts to establish the audio stream between the customer and the rep. If successful, it resolves the promise for the `ACCEPT_CALL` event. With Unified Routing enabled, you must start audio stream establishment as part of the `ACCEPT_CALL` event. If the audio stream connection fails, the telephony provider server triggers the `reroute` API (a server-to-server call). In this case, Salesforce cancels the rep's existing work and reroutes the call to another available rep.
  - Rep misses or declines the call: If the rep doesn't accept the call, Salesforce pushes rep work to a new rep. The call routing process is reinitiated to find the next best available rep.
- Drop the call: When the customer drops the call, the telephony provider server calls the clearRouting API to delete the PSR.

For example, see this sample implementation to support unified routing for inbound voice calls or skills-based routing of voice calls. The `startInboundCall` function handles the initiation of an inbound call and routes it based on configuration settings. It sets the default values for `callInfo` and `flowConfig` if not provided. If no rep is available, it logs an error and returns a rejected promise. Otherwise, it creates a Contact record with a unique ID and a VoiceCall record with the call type, contact, and attributes. If unified routing is enabled, it invokes the `executeOmniFlowForUnifiedRouting` method to handle the call. Otherwise, it publishes a `CALL_STARTED` event. The function returns a promise that resolves with the `callResult` object.

```
startInboundCall(phoneNumber, callInfo, flowConfig) {
        callInfo = callInfo || { isOnHold: false };
        flowConfig = flowConfig || { isUnifiedRoutingEnabled: false };
        callInfo.callStateTimestamp = new Date();
        if (!this.state.agentAvailable) {
            const message = `Rep is not available for a inbound call from phoneNumber -
${phoneNumber}`;
            this.log(message);
            return Promise.reject(new Error(message));
        }
        let callAttributes = { participantType: Constants.PARTICIPANT_TYPE.INITIAL_CALLER
 };
        const id = Math.random().toString(36).substring(5);
        let contact = new Contact({ phoneNumber, id, name: 'Customer '+ id });
        return this.createVoiceCall(undefined, Constants.CALL_TYPE.INBOUND, phoneNumber,
callInfo && callInfo.additionalFields).then((data) => {
            callAttributes.voiceCallId = data.voiceCallId;
            const call = new Call(Constants.CALL_TYPE.INBOUND.toLowerCase(), contact,
callAttributes, new CallInfo(callInfo), data.vendorCallKey || this.generateCallId());
            this.addCall(call);
            const callResult = new CallResult({
                call
            });
          //When Unified Routing is enabled, we need to invoke OmniFlow, otherwise regular
 flow to publish CALL_STARTED event.
            if(flowConfig.isUnifiedRoutingEnabled) {
                console.log('Inside isUnifiedRoutingEnabled ' +
flowConfig.isUnifiedRoutingEnabled);
                var response = this.executeOmniFlowForUnifiedRouting(data, flowConfig);
                console.log('response From execute onmi flow' + response);
            } else {
                console.log('Non UnifiedRouting flow');
              publishEvent({ eventType: Constants.VOICE_EVENT_TYPE.CALL_STARTED, payload:
 callResult });
            }
            return this.executeAsync('startInboundCall', callResult);
        });
    }
```

For example, see this sample `addParticipant` implementation to support unified routing for call transfers. The `addParticipant` function is designed to add a new participant to an ongoing call. The function takes three parameters: contact (the contact to be added), call (the current call object), and isBlindTransfer (a boolean indicating whether the transfer is blind transfer). In the example code, `contact.id` refers to the transfer target ID which can be the Salesforce rep ID or the Salesforce Queue ID. After the `executeOmnifFlow` API call, the remaining transfer flow is the same as that of the inbound flow.

```
async addParticipant(contact, call, isBlindTransfer) {
...
let transferTo = contact.id;
if(this.state?.flowConfig?.isUnifiedRoutingEnabled) {
let callInfoData = {
transferTo,
voiceCallId : newCall.callId
};
let flowConfigData = {
dialedNumber : this.state.flowConfig.dialedNumber
}
await this.executeOmniFlowForUnifiedRouting(callInfoData,flowConfigData);
}
...
}
executeOmniFlowForUnifiedRouting(call, flowConfig) {
...
if(call?.transferTo){
requestObject.transferTarget = call.transferTo;
}
...
return fetch('/api/executeOmniFlow', {
method: 'PATCH',
headers: {
'Content-Type': 'application/json'
},
body: JSON.stringify(requestObject)
}).then(response => response.json()).then((payload) => {
return payload;
}).catch((err) => {
console.log('ERR ',err);
return Promise.reject(err);
});
}
```

SEE ALSO:

Salesforce Object Reference for the Salesforce Platform: ConversationVendorInfo

CHAPTER 13 Transfer Calls

In this chapter ...

This section provides guidelines related to transferring calls.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that’s not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

SEE ALSO:

Call Scenario Diagrams

Configure Estimated Wait Times for Queues

Reps can see estimated wait times for a given queue or skill before transferring a call.

Estimated wait times can be calculated by Salesforce or the telephony provider. By default, estimated wait times are calculated by Salesforce.

Let Salesforce Calculate Estimated Wait Times

To calculate the estimated wait time when routing a call to a queue or skill, at least one call must be accepted within the past 10 minutes. If this criterion isn't met, the estimated wait time field isn't shown.

To let Salesforce calculate estimated wait times:

- Verify the `hasQueueWaitTime` field in the CapabilitiesResult object is set to false. This is the default value.
- Verify the telephony provider queues are mapped to Salesforce queues.
- When accepting a call, provide the telephony provider's `queueId` value to the CallInfo object.
- When accepting a call, optionally provide the telephony provider's `queueTimestamp` and `queueName` value to the CallInfo object.

Let the Telephony Provider Calculate Estimated Wait Times

To display estimated wait times using the telephony provider's calculations, set the `hasQueueWaitTime` field in the CapabilitiesResult object to true.

Optionally include the `queueWaitTime` key in the Contact object that's returned for getPhoneContacts. The `queueWaitTime` field represents the telephony provider's queue wait time measured in seconds.

For example,

```
new Contact ({
  id: 'queueId1'
  type: Constants.CONTACT_TYPE.QUEUE,
  name: "Queue Name",
  queue: 'queueId1',
  queueWaitTime: 240
})
```

SEE ALSO:

Service Cloud Connector API Reference

Knowledge Article: Estimated Wait Time Calculation When Routing Work with Omni-Channel

Salesforce Help: Map Your Salesforce Queues to Telephony Provider Queues

Salesforce Help: Make Smarter Routing Decisions by Checking Service Rep Availability

Agent Availability

This topic provides guidance on agent availability.

Service Cloud Voice for Partner Telephony supports showing agent availability in the Omni-Channel transfer UI. There are two options for showing agent availability.

1. **Vendor-Provided Availability**. If a partner wants to provide availability data as part of the phone book contacts for transfer, they can provide the agent availability as a new availability field on the Contact object. This value is shown in the UI as an availability icon. In order for the Transfer UI to use the vendor-provided availability, the connector also must set `hasAgentAvailability` to `true` in the `AgentConfigResult`.
2. **Salesforce-Provided Availability**. If the partner doesn't provide agent availability as part of the phone book contacts (`hasAgentAvailability` is `false` in `AgentConfigResult`), Salesforce tries to map the phone book contacts to Salesforce agents in the current org and calculates the availability. This agent mapping is done based on the `CallCenterRoutingMap` entries (see Agent Mapping above). If there are matches, Salesforce shows the agent availability for matched agents. The agent available is calculated based on the following criteria:
   - Available—Agent is available for the Voice channel and has 100% percent capacity available
   - Busy—Agent isn't available for the Voice channel
   - Offline—Agent is offline in Omni-Channel

SEE ALSO:

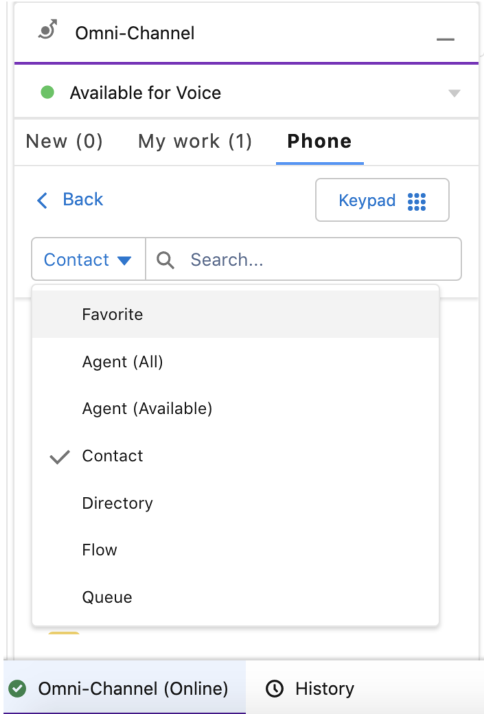
Service Cloud Connector API Reference

Customize the Destination List for Call Transfers in Omni-Channel

Customize the list of transfer destinations to only show the destination types that apply when transferring calls for Omni-Channel.

During an in-progress call, reps can transfer a caller to any of the following transfer destination types through the Omni-Channel widget:

- Agent
- Contact
- Queue
- Flow



By default, the widget displays all transfer destination types, giving reps the option to transfer a call to a specific rep, contact, queue, or flow.

To customize the list so that only certain transfer destination types appear:

- Using the Connector API, set the value of `contactTypes` in `PhoneContactsResult` to the list of transfer destination types you want to appear in the Omni-Channel widget. If `contactTypes` isn't set, all transfer destination types are listed in the widget for reps to pick from.

SEE ALSO:

Service Cloud Connector API Reference

Enable Voice Call Transfers Using Omni-Channel Flows and Partner Telephony

Configure this feature to enable voice call transfers via Salesforce Omni-Channel flows.

This configuration applies to the Service Cloud Voice with Partner Telephony telephony model. To enable this feature for Service Cloud Voice with Amazon Connect or Service Cloud Voice for Partner Telephony from Amazon Connect, go to Enable Voice Call Transfers Using Omni-Channel Flows and Amazon Connect.

Omni-Channel flows can be used to transfer voice calls through External Routing. Configure this feature to enable voice call transfers using Omni-Channel flows. When this feature is enabled, all active flows of process type Omni-Channel Flow that are assigned to the phone channel appear in the Omni-Channel widget for reps to select as transfer destinations.

To enable voice call transfers using Omni-Channel flows:

1. Verify that your system is configured to Create a Voice Call Record for transfer.
2. While calling the Execute an Omni-Channel Flow REST API, verify that your system is properly routing the voice calls. The `flowDevName` parameter is required. Don't set the `dialedNumber` parameter.
3. Verify that the `hasTransferToOmniFlow` parameter in CapabilitiesResult is set to `true`.

When a rep transfers a call to an Omni-Channel flow, Salesforce calls the addParticipant Connector API along with the fully qualified name of the Omni-Channel flow. For example, `{Namespace}__{API Name of the Omni-Channel Flow}`.

The telephony partner system creates a voice call record for transfer through the Create a Voice Call Record REST API, and then executes the Omni-Channel flow through the Execute an Omni-Channel Flow REST API. A sample implementation of this entire process can be found in the vendor-sdk.js script of the demo connector in GitHub.

SEE ALSO:

Service Cloud Connector API Reference

Transfer Calls to a Queue

Transfer a call to a Salesforce queue so that a supervisor can see the transferred call waiting in the queue.

To use this feature, when transferring a call with the Create a Voice Call Record Telephony API, set `initiationMethod` to `"Transfer"` and set `queue` to either the Salesforce queue object ID or the vendor's external queue ID. See Create a Voice Call Record.

Perform a Blind Transfer

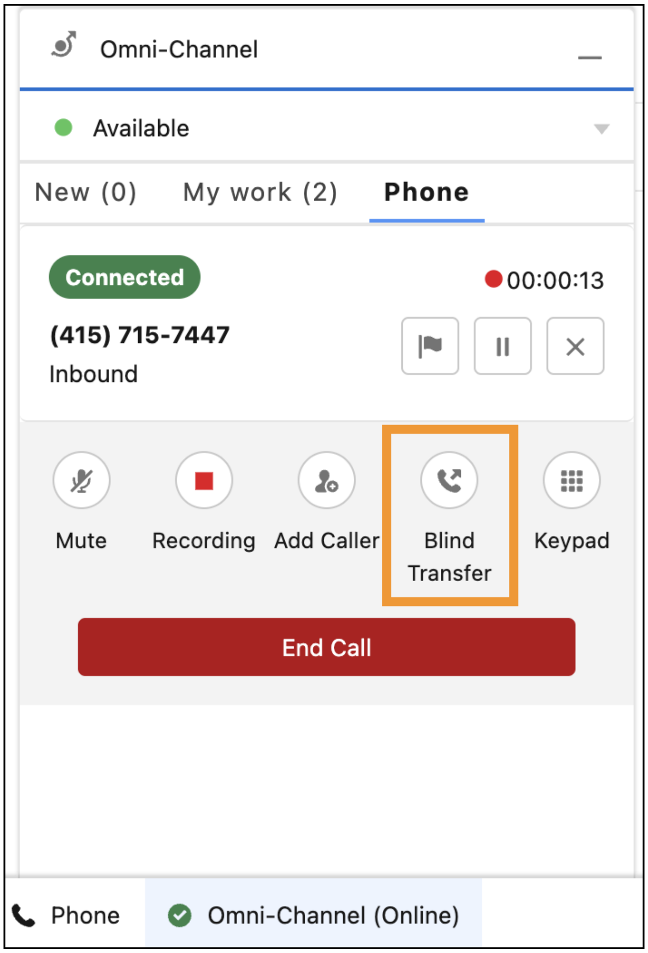
With the blind transfer feature, vendors can use the addParticipant Connector API method to hang up and transfer a call rather than add a caller to an existing conversation.

To enable blind transfer:

1. Using the Connector API, set the value of `hasBlindTransfer` in `CapabilitiesResult` to `true`. See `getCapabilities()`.
2. Update the implementation of `addParticipant()` to perform a blind transfer when the `isBlindTransfer` argument is true. A sample blind transfer is implemented in the demo connector on page 32.

To perform a blind transfer:

1. Using a softphone, when the call is connected, a new button with the label "Blind Transfer" shows up. Click this button to call `addParticipant(contact, call, true)`.
2. Using a hardphone, perform a blind transfer and publish the `PARTICIPANT_ADDED` event.



SEE ALSO:

Service Cloud Connector API Reference

Use Click-to-Dial for Transfers

Enable click-to-dial for phone numbers so that a rep can call or transfer to that number.

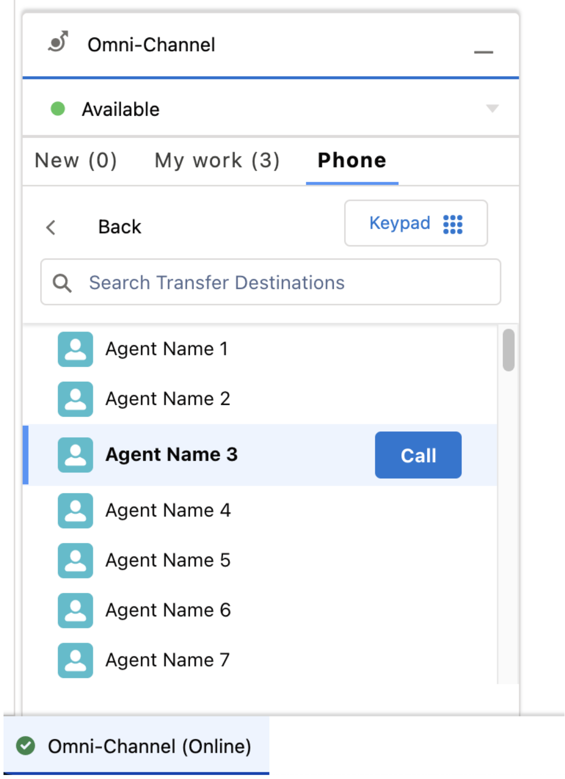
Phone numbers (using the lightning:clickToDial Lightning component) are clickable when a rep isn't on an active call. Clicking the phone number starts an outbound call. A rep can also transfer a call when a call is in progress and the phone number is clicked. Clicking a click-to-dial component transfers the call to the new phone.

When clicking a phone number, the rep is prompted with two options. The rep can either add the phone number to the call or use a blind transfer to transfer the call without context. To support a blind transfer, see Perform a Blind Transfer.

Note: Phone numbers no longer support click-to-dial when the call ends or when a participant has already been added to the call.

Phone Contact Search

When a rep adds a participant to a call, Salesforce provides a UI to search for transfer destinations.



The contacts are returned from the vendor when implementing `getPhoneContacts()`.

The API supports a long list of contacts and lets reps search for contacts that they want to transfer to.

Search and Pagination

`getPhoneContacts()` is called by Salesforce with a filter argument. The filter object has the keys `contains`, `limit`, and `offset`. `limit` is a constant set to 50 and `offset` is calculated by the UI.

If the vendor supports contact search (by specifying `AgentConfigResult.hasContactSearch` as `true`), the following events occur.

- When the transfer UI is shown, `getPhoneContacts({limit: 50})` is called and the vendor can return the most recent 50 items.
- With every search or filtering request, Salesforce calls `getPhoneContacts({contains: string, limit: 50, offset: number }).`
- When scrolling down ("progressive loading"), Salesforce calls `getPhoneContacts({contains: string, filter: string, limit: 50, offset: calculated}).`

Example scenarios:

- At initialization time: `getPhoneContacts({limit: 50}) → [a1,b2... x50]` (most recent contacts)
- When searching for "xyz": `getPhoneContacts({contains: "xyz", limit: 50, offset: 0}) → [xyz1,xyz2... xyz50]`
- When scrolling down: `getPhoneContacts({contains: "xyz", offset: 50, limit: 50}) → [xyz51,xyz52... xyz100]`

Pseudo Contact Search

A new capability called `hasContactSearch` is available. When the vendor doesn't implement contact search, the capability is set to `false` or undefined. If the capability is `false` or undefined, Salesforce gets all phone contacts (up to 1000) and stores them in memory. Then, for every scroll or search, Salesforce does a "pseudo search," or a simple filtering, and shows the data that matches the filter. `getPhoneContacts()` is called one time.

SEE ALSO:

Service Cloud Connector API Reference

CHAPTER 14 Disable Call Actions

You can disable call handling options such as end call, dial pad, and phone book in the softphone to support compliance with internal policies and industry regulations.

To disable the dial pad, when implementing getVoiceCapabilities(), set the value of the isDialPadDisabled field of VoiceCapabilitiesResult to true.

To disable the phone book, when implementing getVoiceCapabilities(), set the value of the isPhoneBookDisabled field of VoiceCapabilitiesResult to true.

To disable the End Call button, when implementing getVoiceCapabilities(), set the value of the endCallDisabled field of CallInfo to true.

The CALL_UPDATED event is used to send additional information during a call. This event uses the CallResult class object and so you can use the CallInfo object to update the softphone controls. This event can be published during the call.

SEE ALSO:

VoiceCapabilitiesResult

CallInfo

publishEvent

CHAPTER 15 Desk Phone Support

If the telephony provider supports desk phones, reps can make outbound calls or answer inbound calls from their desk phone. Reps can also decline calls and initiate transfers from their desk phone. By default, the softphone is enabled for all reps. To use a deskphone, reps should enable the deskphone from the Omni-Channel utility.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Test Device Selection API

The Service Cloud Connector API `getAgentConfig()` and `setAgentConfig()` methods should be implemented in the vendor connector. The `getAgentConfig()` method is invoked when the rep clicks the Agent Settings icon. It returns an `AgentConfigResult` object. For example:

```
new AgentConfigResult({
    phones: ["SOFT_PHONE", "DESK_PHONE"],
    selectedPhone: new Phone("DESK_PHONE", “5554443333”)
})
```

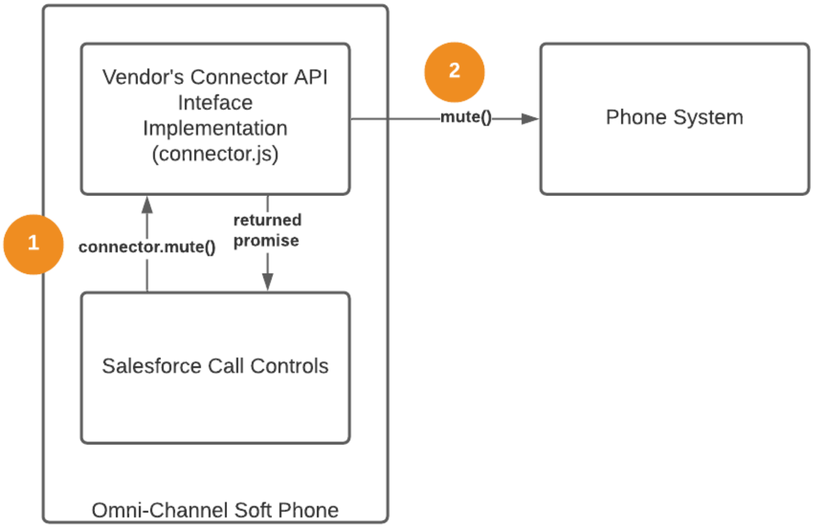
The `setAgentConfig()` method is implemented by the vendor and returns a `SetAgentConfigResult` object indicating whether the operation is successful. For example:

```
new SetAgentConfigResult({ success: true })
```

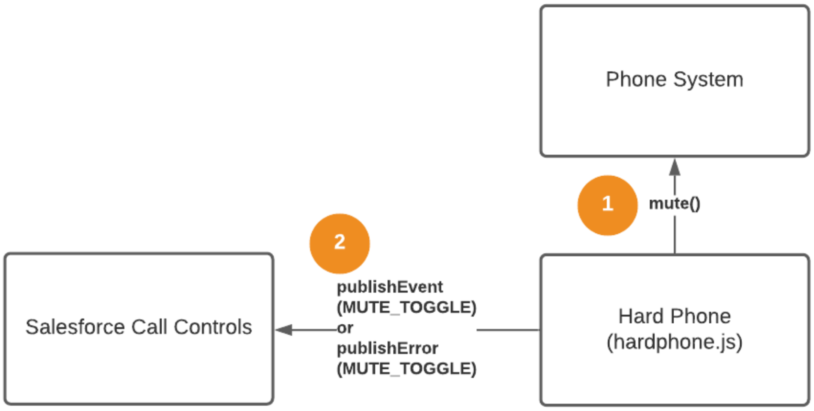
Publish an Event from a Desk Phone

When using the Omni-Channel softphone, call control events and errors get published automatically by Salesforce. Each promise can be resolved or rejected from the vendor's Connector API Interface implementation. However, when using a desk phone, events and errors must be published from the vendor's code using `publishEvent()` and `publishError()`.

For example, when clicking the mute button from a softphone, the `mute()` method is called on the vendor's Connector API Interface implementation and a promise is sent back to Salesforce.



When using a hard phone, you must explicitly call `publishEvent()` when the mute occurs.


Phone System
1
mute()
2
publishEvent (MUTE_TOGGLE)
or
publishError (MUTE_TOGGLE)
Salesforce Call Controls
Hard Phone (hardphone.js)

The `publishEvent()` method can be called from the Connector API Interface implementation with many potential event types. The `publishError()` method can be called for desk phone events for the same events. For example:

- MUTE_TOGGLE — when the phone is muted or unmuted
- HOLD_TOGGLE — when the phone is on hold or resumed
- RECORDING_TOGGLE — when the recording is enabled or disabled
- SWAP — when the hold state of two callers is switched
- CONFERENCE — when two calls are being conferenced
- ADD_PARTICIPANT — when adding a call participant

See the reference documentation for all the possible event types.

Accept or Decline a Call

If accepting or declining a call from a hard phone (desk phone), add the attribute `isSoftphoneCall: false` to `call.callInfo` when calling the `publishEvent()` method.

```
publishEvent({
  eventType: Constants.VOICE_EVENT_TYPE.CALL_CONNECTED,
  payload: new CallResult({ call }) // "call" contain the PhoneCall
object
});
```

When initiating an outbound or transfer call from a desk phone, `isSoftphoneCall` should also be `false`.

Disable Call Controls Dynamically

When publishing call events such as `CALL_STARTED` using `publishEvent()`, the connector can disable call controls if you prefer that reps not use the Omni-Channel softphone call controls at a desk phone. By default, all controls are enabled, but you can pass any of these values in the `callInfo` parameter (from the `PhoneCall` object) to disable call controls.

```
callInfo: {
  acceptEnabled: boolean,
  declineEnabled: boolean,
  holdEnabled: boolean,
  muteEnabled: boolean,
  extensionEnabled: boolean,
  swapEnabled: boolean,
  conferenceEnabled: boolean,
  extensionEnabled: boolean,
  recordEnabled: boolean,
  addCallerEnabled: boolean
}
```

Start an Outbound Call Programmatically

While in most cases, reps start outbound calls by click-to-dial on a contact or the Omni-Channel dial pad, there's also a way to start a call programmatically using the Service Cloud Connector API. Specifically, use the `publishEvent()` method with the `CALL_STARTED` event type and a `CallResult` payload.

This code sample is adapted from the Demo Connector on page 32 in GitHub (byo-demo-connector).

```
/**
 * Start a call
 * @param {Contact} contact
 * @param {Object} callInfo (callInfo.isSoftphoneCall is false if
dialing from a desk phone)
 */
function dial(contact, callInfo) {
  const callAttributes = { participantType:
```

```
Constants.PARTICIPANT_TYPE.INITIAL_CALLER };
  const call = new Call(Constants.CALL_TYPE.OUTBOUND, contact,
callAttributes, new CallInfo(callInfo));
  if (!callInfo.isSoftphoneCall) {
   publishEvent({ eventType: Constants.VOICE_EVENT_TYPE.CALL_STARTED,
 payload: new CallResult({ call })});
  }
}

/**
 * Example usage
 */
dial(new Contact({ phoneNumber: "5554445555"}), {
  isSoftphoneCall: false,
  callStateTimestamp: new Date(),
  isOnHold: false,
  isMuted: false,
  isRecordingPaused: false,
  muteEnabled: true,
  swapEnabled: true,
  conferenceEnabled: true,
  extensionEnabled: true,
  holdEnabled: true,
  recordEnabled: true,
  addCallerEnabled: true
});
```

SEE ALSO:

- *Service Cloud Connector API Reference*
- *Service Cloud Connector API Reference*: Get Started with the Service Cloud Connector API
- *Service Cloud Connector API Reference*: publishEvent
- *Service Cloud Connector API Reference*: GenericResult
- *Service Cloud Connector API Reference*: getAgentConfig
- *Service Cloud Connector API Reference*: setAgentConfig
- *Service Cloud Connector API Reference*: CallResult
- *Service Cloud Connector API Reference*: PhoneCall

CHAPTER 16 Enable Headset Support

Enable headset support so that reps can control call actions such as accept, mute, unmute, and decline calls from their headsets.

This feature applies to the following telephony model:

- Service Cloud Voice with Partner Telephony

Perform the following steps using the Connector API to enable headset support.

- When implementing getVoiceCapabilities(), set the value of isHidSupported in VoiceCapabilitiesResult to true. When the rep selects the headset, the headset information is sent using the SET_AGENT_CONFIG event.
- Add handlers to listen to the events from the headset, adding parsers for each type of HID device to determine the event type as accept, mute, unmute, or decline.
- Each headset model has a specific signal for each of the actions. Configure the Connector API interface to explicitly publish each headset call event and error using the publishEvent() and publishError() APIs, respectively. For example, when muting the headset, call the mute() method on the Connector API Interface implementation and send a promise back to Salesforce.

See a sample implementation of the headset capabilities in the demo connector. The demo connector code available in Github is not generic which means a headset parser is required for each kind of headset, for example the Github sample supports only certain models such as:

- Plantronics Blackwire 5220 Series - accept/decline/mute/unmute call actions supported
- Jabra Evolve Link Ms - accept/decline call actions supported

SEE ALSO:

publishEvent

CHAPTER 17 Additional Info

In this chapter ...

This section provides additional information about your Service Cloud Voice for Partner Telephony implementation.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Einstein Conversation Insights (Call Coaching)

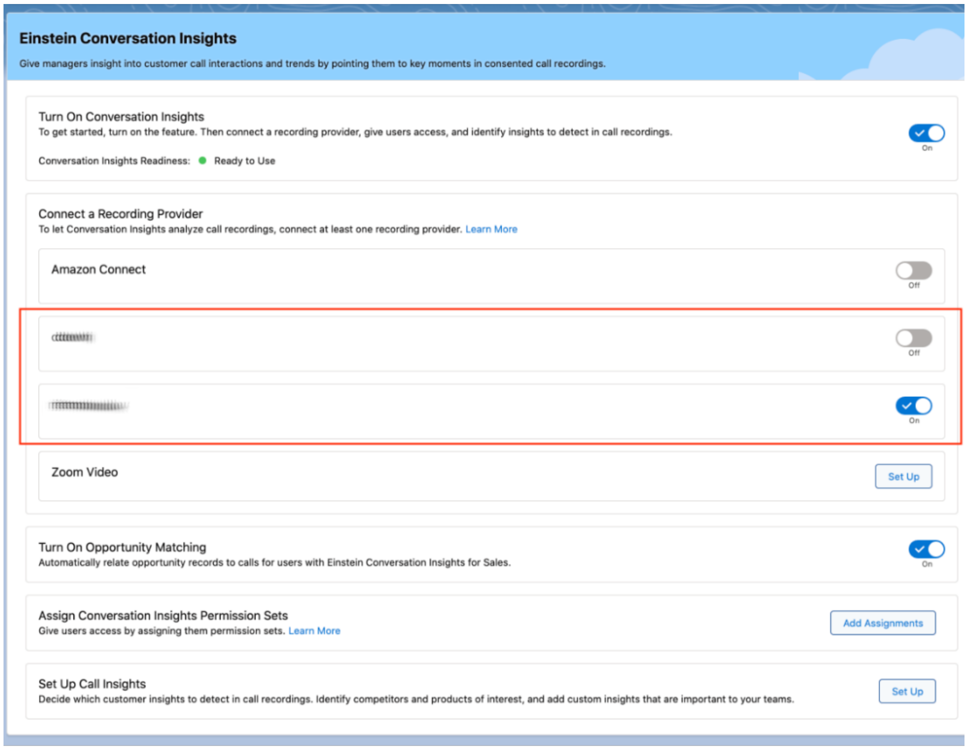
Provide support for Einstein Conversation Insights to Service Cloud Voice reps.

In order to support Einstein Conversation Insights (ECI) for customers, vendors must perform several steps.

ECI is available in Performance and Unlimited Editions, and as an add-on in Enterprise Edition. Please reach out to your Salesforce account executive if you require one of these orgs to test your ECI integration.

For more information about setting up ECI, see Set Up Einstein Conversation Insights in Salesforce Help.

1. Turn on this feature in the ConversationVendorInfo object by specifying the `einsteinConversationInsightsSupported` value to `true`. This value creates an entry in the General Settings section of Conversation Insights, which the admin can choose to toggle on or off.



Einstein Conversation Insights

Give managers insight into customer call interactions and trends by pointing them to key moments in consented call recordings.

Turn On Conversation Insights
To get started, turn on the feature. Then connect a recording provider, give users access, and identify insights to detect in call recordings.
Conversation Insights Readiness: ● Ready to Use
On

Connect a Recording Provider
To let Conversation Insights analyze call recordings, connect at least one recording provider. Learn More

Amazon Connect — Off

[redacted] — Off

[redacted] — On

Zoom Video — Set Up

Turn On Opportunity Matching
Automatically relate opportunity records to calls for users with Einstein Conversation Insights for Sales.
On

Assign Conversation Insights Permission Sets
Give users access by assigning them permission sets. Learn More
Add Assignments

Set Up Call Insights
Decide which customer insights to detect in call recordings. Identify competitors and products of interest, and add custom insights that are important to your teams.
Set Up

2. Implement `service_cloud_voice.RecordingMediaProvider` on page 125 in the Apex class. The method `getSignedUrls` is called by Salesforce with named credentials (if set up for the org) and a list of vendor call keys. The named credentials can be used to make a call out to the partner system for authentication. The partner has to return a signed URL for each vendor call key (or an error) that is valid for at least five minutes. Salesforce then uses this signed URL to download the recording file for that particular voice call and analyze it.

SEE ALSO:

Set Up Service Cloud Voice for Partner Telephony in Your Org

Replay Active Calls on Refresh

When a rep refreshes the Salesforce page in the middle of a call, `getActiveCalls()` is called to restore the state of the VoiceCall record page and to attempt to replay the call with the same payload.

Note: This section assumes that you've implemented the `getActiveCalls()` API in the connector.

For example, if an active call is refreshed while it's connected, a CALL_CONNECTED event is replayed with the same call. If an active call is refreshed while it's ringing, a CALL_STARTED event is replayed with the same call.

The `getActiveCalls()` method is implemented in the demo connector on page 32 using JS local storage, but it's best that you implement it on the server side rather than in local storage. We recommend storing the active calls in a server-side database as soon as a call starts, and updating the database on subsequent events. This way, Salesforce receives the most current data when `getActiveCalls()` is called.

In case you want to replay the calls yourself, we recommend that you set the `isReplayable` field in `callInfo` attribute of the call object to `false` so that the base connector doesn't replay the calls when the rep becomes available.

```
getActiveCalls() {
   // Get the current calls in progress
   const callsInProgress = getCallsInProgress();

   // Create active calls and set isReplayable to false
    const activeCalls = callsInProgress.map((call) => {
        call.callInfo = new CallInfo({ isReplayable: false });
        return new PhoneCall(call);
    });

    // Return the active calls back in ActiveCallsResult
    const activeCallsResult = new ActiveCallsResult({ activeCalls });
    return activeCallsResult;
}
```

SEE ALSO:

Service Cloud Connector API Reference

Service Cloud Connector API Reference: getActiveCalls

Host the Connector as a Visualforce Page

You can also host your connector as a Visualforce page in Salesforce and package it.

1. Inside your connector, call this command:

```
// Generates files at dist/ for the connector
$ npm run build:dev
// (you can use npm run build:prod to generate minified js files)
```

2. Add these files to `main/default/staticresources` in the Salesforce DX project.
3. Make sure you have the corresponding `-meta.xml` files for the resources added in `staticresources`. For help adding static resources, see Salesforce DX Project Structure and Source Format.
4. Create a connector Visualforce page inside `scv-external-telephony-quickstart/force-app/main/default/pages/` as follows:

```
<apex:page>
<apex:includeScript value="{!$Resource.REPLACE_WITH_CONNECTOR_RESOURCE_NAME}"/>
</apex:page>
```

5. Update adapterUrl and reqAdapterUrl to /apex/`<namespace>`__`<connector visual force page name>` in the ConversationVendorInfo file of your Salesforce DX project.
6. Create and install a new package version.

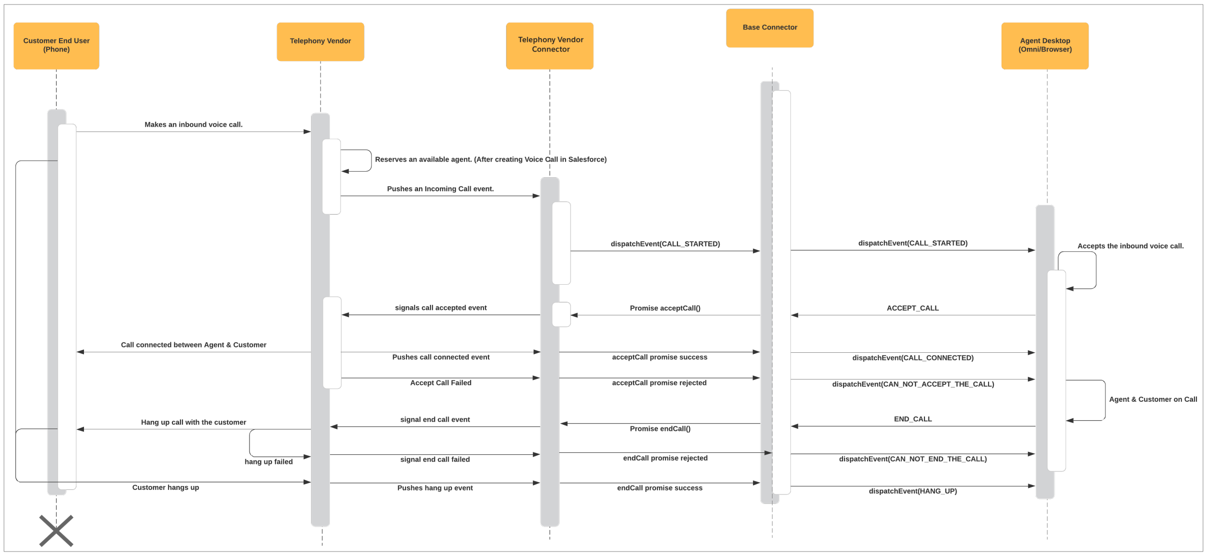
Note: For different calls to SCRT2 such as transcription, call recording, and create voice call (salesforce-scrt.com/telephony/v1), you need a standalone server which can receive requests from the Visualforce page connector and call SCRT2 using JWT.

Alternatively, you can use the Apex web service. The Visualforce page connector can call the Apex web service, which can call the SCRT2 endpoints with a valid JWT token for voice call creation and transcription. For information on Apex Web Services, see Exposing Apex Classes as REST Web Services.

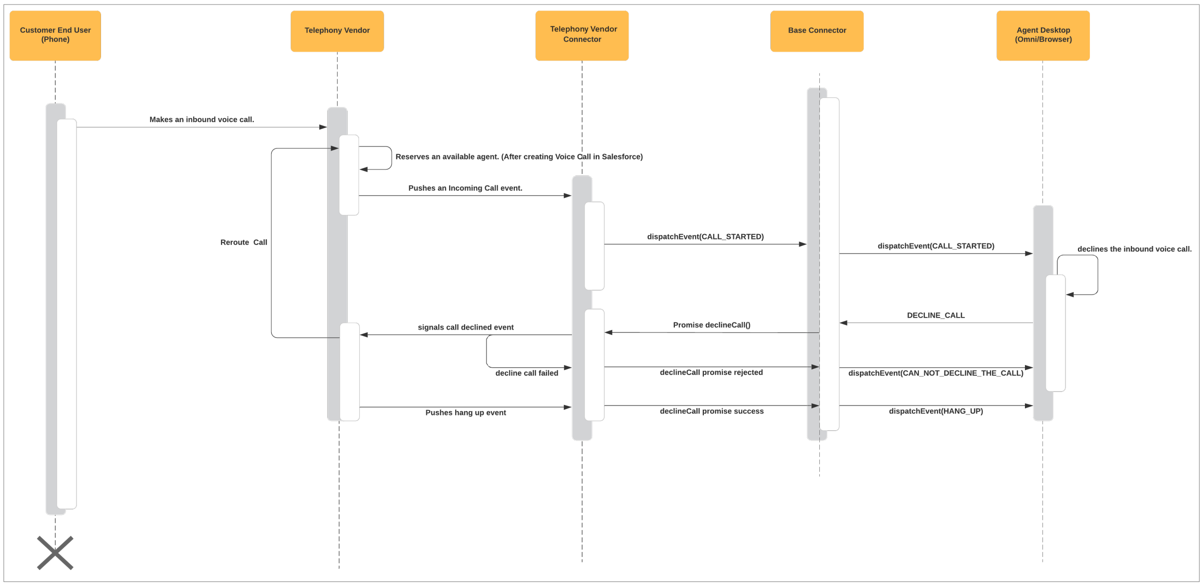
Call Scenario Diagrams

Understand different call scenarios, such as a rep receiving or declining an inbound call.

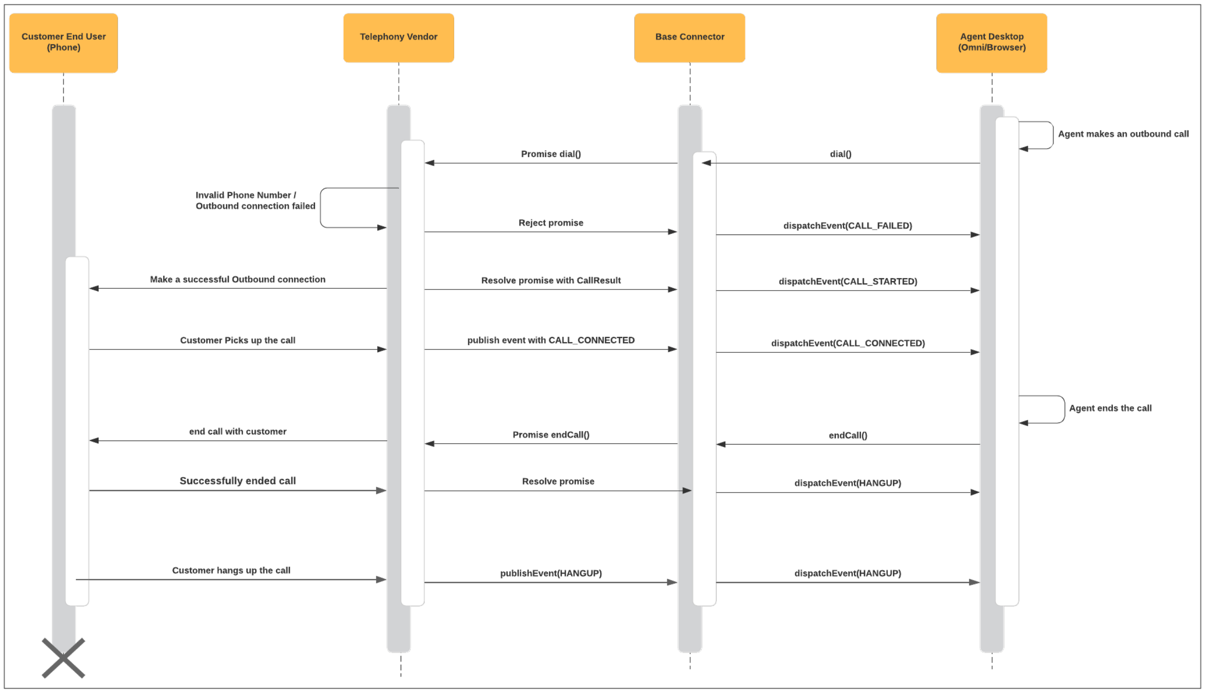
Rep Accepts Inbound Call


Customer End User (Phone)
Telephony Vendor
Telephony Vendor Connector
Base Connector
Agent Desktop (Omni/Browser)
Makes an inbound voice call.
Reserves an available agent. (After creating Voice Call in Salesforce)
Pushes an Incoming Call event.
dispatchEvent(CALL_STARTED)
dispatchEvent(CALL_STARTED)
Accepts the inbound voice call.
signals call accepted event
Promise acceptCall()
ACCEPT_CALL
Call connected between Agent & Customer
Pushes call connected event
acceptCall promise success
dispatchEvent(CALL_CONNECTED)
Accept Call Failed
acceptCall promise rejected
dispatchEvent(CAN_NOT_ACCEPT_THE_CALL)
Agent & Customer on Call
Hang up call with the customer
signal end call event
Promise endCall()
END_CALL
hang up failed
signal end call failed
endCall promise rejected
dispatchEvent(CAN_NOT_END_THE_CALL)
Customer hangs up
Pushes hang up event
endCall promise success
dispatchEvent(HANG_UP)

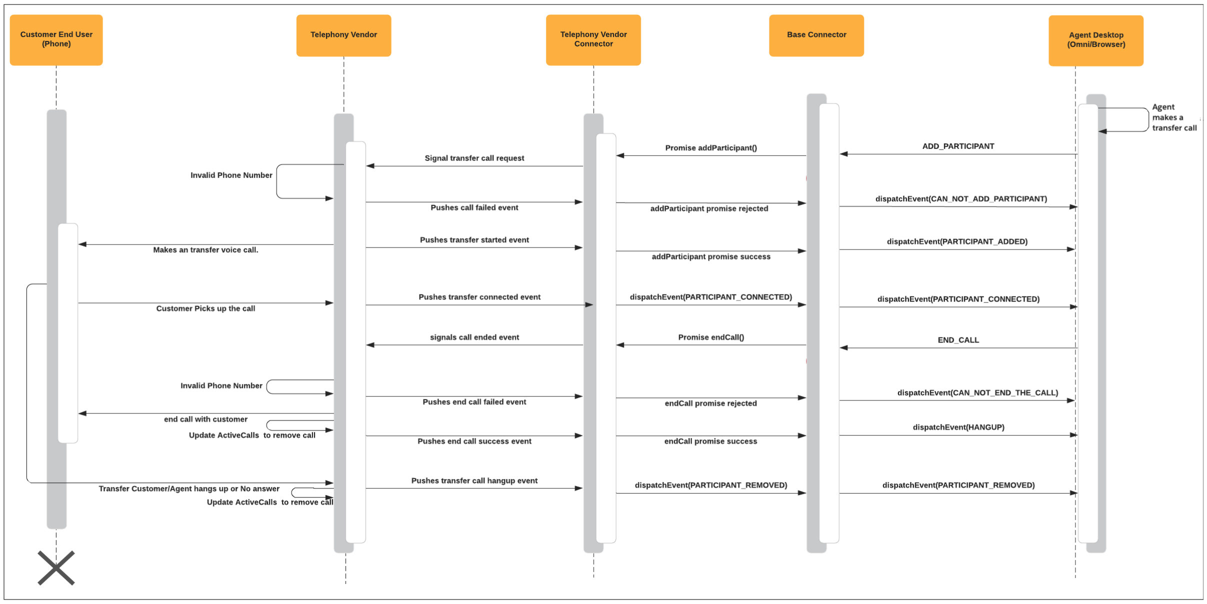
Rep Declines Inbound Call


Customer End User (Phone)
Telephony Vendor
Telephony Vendor Connector
Base Connector
Agent Desktop (Omni/Browser)
Makes an inbound voice call.
Reserves an available agent. (After creating Voice Call in Salesforce)
Pushes an Incoming Call event.
Reroute Call
dispatchEvent(CALL_STARTED)
dispatchEvent(CALL_STARTED)
declines the inbound voice call.
DECLINE_CALL
Promise declineCall()
signals call declined event
decline call failed
declineCall promise rejected
dispatchEvent(CAN_NOT_DECLINE_THE_CALL)
Pushes hang up event
declineCall promise success
dispatchEvent(HANG_UP)

Rep Makes Outbound Call


Customer End User (Phone)
Telephony Vendor
Base Connector
Agent Desktop (Omni/Browser)
Agent makes an outbound call
Promise dial()
dial()
Invalid Phone Number / Outbound connection failed
Reject promise
dispatchEvent(CALL_FAILED)
Make a successful Outbound connection
Resolve promise with CallResult
dispatchEvent(CALL_STARTED)
Customer Picks up the call
publish event with CALL_CONNECTED
dispatchEvent(CALL_CONNECTED)
Agent ends the call
end call with customer
Promise endCall()
endCall()
Successfully ended call
Resolve promise
dispatchEvent(HANGUP)
Customer hangs up the call
publishEvent(HANGUP)
dispatchEvent(HANGUP)

Rep Transfers a Call


Customer End User (Phone)
Telephony Vendor
Telephony Vendor Connector
Base Connector
Agent Desktop (Omni/Browser)
Agent makes a transfer call
ADD_PARTICIPANT
Promise addParticipant()
Signal transfer call request
Invalid Phone Number
Pushes call failed event
addParticipant promise rejected
dispatchEvent(CAN_NOT_ADD_PARTICIPANT)
Makes an transfer voice call.
Pushes transfer started event
addParticipant promise success
dispatchEvent(PARTICIPANT_ADDED)
Customer Picks up the call
Pushes transfer connected event
dispatchEvent(PARTICIPANT_CONNECTED)
dispatchEvent(PARTICIPANT_CONNECTED)
signals call ended event
Promise endCall()
END_CALL
Invalid Phone Number
Pushes end call failed event
endCall promise rejected
dispatchEvent(CAN_NOT_END_THE_CALL)
end call with customer
Update ActiveCalls to remove call
Pushes end call success event
endCall promise success
dispatchEvent(HANGUP)
Transfer Customer/Agent hangs up or No answer
Update ActiveCalls to remove call
Pushes transfer call hangup event
dispatchEvent(PARTICIPANT_REMOVED)
dispatchEvent(PARTICIPANT_REMOVED)

Line-Specific Controls with Service Cloud Voice for Partner Telephony

We now support enabling and disabling the Remove Participant UI control for each participant during a conference call. For example, you can disable the Remove Participant UI control for the primary caller.

The Remove Participant button can be disabled by setting the `removeParticipantVariant` parameter of the Service Cloud Connector API `CallInfo` object.

Possible values are:

- `ALWAYS`: Remove participant button is always enabled.
- `NEVER`: Remove participant button is always disabled.
- `ALWAYS_EXCEPT_ON_HOLD`: Remove participant button is disabled when the participant is on hold.

SEE ALSO:

Service Cloud Connector API Reference

Download Connector Logs

Partners can log messages from their connector and then download these logged messages as a text file.

To enable logging, set `debugEnabled` to true in the `CapabilitiesResult` object that is returned as a result of the `getCapabilities()` method in the connector interface.

When `debugEnabled` is set to true, Settings in Omni-Channel has a new link titled “Download agent debug information” under Debugging. Click this link to download a text file that contains the data logged.

To log a message in the connector:

```
import { log, Constants } from 'scv-connector-base';
```

Example for logging an error message:

```
log({message: "Log this error"}, Constants.LOG_LEVEL.ERROR);
```

Example for logging an info message:

```
log({anyobject: 1}, Constants.LOG_LEVEL.INFO);
```

CHAPTER 18 Service Cloud Voice for Partner Telephony Apex Reference

Service Cloud Voice for Partner Telephony uses several Apex classes.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

To automate all user operations on the contact center in Salesforce, partners should provide an Apex integration class that implements the supported Apex interfaces.

Note: Make sure, along with implementing the supported interfaces, you also declare that capability in ConversationVendorInfo. If you don't declare the capability, your implemented methods aren't called.

When implementing an Apex interface method, if the operation is successful, create a response object using the response constructor with success as `true` and set the response field. If the operation fails, create a response object with success as `false`, an appropriate `errorMessage`, and a null response field.

The following interfaces can be implemented by partners.

service_cloud_voice.ContactCenterInfo

For guidance on this Apex class, see Send Real-Time Signals on page 81.

service_cloud_voice.IntelligenceServiceAndSignalsInfo

For guidance on this Apex class, see Send Real-Time Signals on page 81.

service_cloud_voice.IntelligenceServiceProvider

For guidance on this Apex interface, see Send Real-Time Signals on page 81.

service_cloud_voice.IntelligenceServiceRequest

For guidance on this Apex class, see Send Real-Time Signals on page 81.

service_cloud_voice.IntelligenceServiceResponse

For guidance on this Apex class, see Send Real-Time Signals on page 81.

service_cloud_voice.IntelligenceSignalType

For guidance on this Apex class, see Send Real-Time Signals on page 81.

service_cloud_voice.GroupSetup

For guidance on this interface, see Associate Partner Telephony Users and Groups with Queues.

service_cloud_voice.IntelligenceServiceProvider

For guidance on this interface, see Enable Conversation Intelligence for Partner System Intelligence Signals.

service_cloud_voice.KeyProvider

Implement this interface to automate key provisioning and renewal. Whenever a Salesforce contact center is connected to the partner system (during contact center creation flow or later using the **Connect Account** button on the contact center details page), Salesforce calls the `getPublicKey` method on the integration class. The same method is also called when the public key is renewed (using the **Update Key** button on the contact center details page).

For this interface, turn on the `keyProvisioningSupported` capability.

```
KeyResponse getPublicKey(ContactCenterInfo contactCenterInfo);
```

`KeyResponse` contains four fields: a success flag, an error message, a public key, and the expiration date of the public key.

service_cloud_voice.PartnerConnector

For this interface, turn on the `namedCredentialSupported` capability. This interface contains one method.

```
ConnectPartnerResponse connect(service_cloud_voice.ContactCenterInfo
 contactCenterInfo);
```

The `ConnectPartnerResponse` response has three fields:

- `Success`: Boolean value for whether operation was a success or failure.
- `contactCenters`: Map of partner contact center ID and partner contact center name.
- `errorMessage`: Failure message if the operation failed.

Use the following constructor to create a response instance:

```
ConnectPartnerResponse(boolean success, Map<String, String>
contactCenters, String errorMessage);
```

The `ContactCenterInfo` input value contains information about the contact center, such as the internal name, the display name, the org ID, the partner contact center ID, and the fully qualified name of the named credential selected by user.

service_cloud_voice.PartnerSSO

Implement this interface to set up SSO for the reps with Salesforce as an identity provider. See Set Up Single Sign-On on page 40 for more information.

For this interface, turn on the `agentSSOSupported` capability. It contains two methods.

```
PartnerResponse setupSamlIdentityProvider(SetupSamlIdpRequest
setupSamlIdpRequest);
```

This method is used to create a SAML identity provider in the partner account. `SetupSamlIdpRequest` has two fields: the SAML XML and the named credential. The SAML XML is generated from the Salesforce identity provider. You can reuse an identity provider since this is needed one time per account.

```
ConnectedAppSetupParams getConnectedAppSetupParams(ContactCenterInfo
 contactCenterInfo);
```

This method is called from Salesforce to get the `ConnectedAppSetupParams`, which contains fields to create a connected app in Salesforce. `ConnectedAppSetupParams` contains `acsUrl`, `entityUrl`, `customAttributes` and `sloUrl`.

service_cloud_voice.PhoneNumberProvider

Implement this interface to support listing phone numbers when creating contact center channels.

```
PhoneNumberResponse
listPhoneNumbers(service_cloud_voice.PhoneNumberRequest
phoneNumberRequest);
```

`PhoneNumberResponse` contains a list of `PhoneNumberInfos`. `PhoneNumberInfo` contains phoneNumber, countryCode, and an identifier.

`PhoneNumberRequest` contains `ContactCenterInfo`.

service_cloud_voice.QueueManager

For guidance on this interface, see Associate Partner Telephony Users and Groups with Queues.

service_cloud_voice.QueueSetup

For guidance on this interface, see Associate Partner Telephony Users and Groups with Queues.

service_cloud_voice.RecordingMediaProvider

For this interface, turn on the `einsteinConversationInsightsSupported` capability. It contains one method to provide the recording URLs, which can be downloaded and analyzed.

```
RecordingMediaResponse
getSignedUrls(service_cloud_voice.RecordingMediaRequest request);
```

The `RecordingMediaResponse` response is a list of `RecordingMediaItem` objects. Each item contains `recordingUrl`, `partnerVoiceCallId`, and an error message if the recording URL isn't present. Use the following constructor to create an instance of `RecordingMediaItem`:

```
RecordingMediaItem(String vendorCallKey, String signedRecordingUrl,
String expiryTime, String errorCode);
```

Set the `expiryTime` parameter to the length of time, measured in minutes, before the signed recording URL (`signedRecordingUrl`) expires. If `signedRecordingUrl` doesn't expire, set `expiryTime` to `NULL`. Set the `errorCode` parameter to the HTTP error code that's returned if the recording URL (`recordingUrl`) doesn't exist. Salesforce will retry the voice call for all error codes except for error code 404. If an error code is not expected, set the value to `NULL`.

The `RecordingMediaRequest` input value contains the named credentials to be used for the callout, and the list of partner Voice Call IDs for the recording URLs.

service_cloud_voice.TransferDestinationProvider

For this interface, turn on the `partnerTransferDestinationsSupported` capability. It contains one method to fetch rep queues.

```
TransferDestinationResponse
getQueues(service_cloud_voice.ContactCenterInfo contactCenterInfo);
```

The `TransferDestinationResponse` response contains three fields: a success flag, a map of queue ID and queue names, and an error message for a failed operation. Use the following constructor to create a response instance:

```
TransferDestinationResponse (boolean success, Map<String, String>
queues, String errorMessage);
```

service_cloud_voice.UserSyncing

Implement this interface to automate user syncing. Whenever a user is added or removed from the Salesforce contact center, Salesforce calls these methods on the integration class. For this interface, turn on the `agentSSOSupported` capability.

```
UserSyncingResponse addUsersToContactCenter(UserSyncingRequest
userSyncingRequest);
```

The `UserSyncingResponse` response contains three fields: a success flag, an error message, and a map of a Salesforce user ID and a partner system user ID.

Note: User addition and removal is atomic. That is, users are added or removed in Salesforce only if all the users in that batch are successfully added or removed in a partner system.

`UserSyncingRequest` contains a Salesforce contact center ID, a named credential, and a list of UserInfo objects, each having a Salesforce user ID, first name, last name, and the Salesforce username of the rep.

service_cloud_voice.UpdateOrgDomainProvider

Your org's My Domain is a subdomain for the URLs that Salesforce uses to serve your org. That means that some URLs that are used for Service Cloud Voice features, such as the Connector URL, the telephony API (SCRT2 URL), and the Connect API URL, contain the domain value. If you change your My Domain name, that subdomain value changes and impacts the URLs used by Service Cloud Voice features. In that situation, this Apex class notifies you of updated domain information.

Implement this interface to get notified of My Domain changes in your org.

```
PartnerResponse
updateOrgDomainValues(service_cloud_voice.ContactCenterInfo
contactCenterInfo);
```

This method has a parameter, `ContactCenterInfo`, which contains two new properties:

- `orgDomainVal` contains the latest value for the My Domain URL.
- `scr2Url` contains the URL for the SCRT2 server.

`ContactCenterInfo` also has the following properties: `contactCenterId`, `partnerContactCenterId`, `internalName`, `displayName`, `namedCredentials`, `orgId`.

CHAPTER 19 Service Cloud Connector API Reference

Pass information between your partner telephony, Messaging, or Contact Center as a Service (CCaaS) system and a Salesforce org by using the Service Cloud Connector API.

SEE ALSO:

Service Cloud Connector API Reference

CHAPTER 20 Service Cloud Voice for Partner Telephony Troubleshooting

Review tips for troubleshooting common problems.

Important: This guide is for telephony providers who are creating a solution that integrates Service Cloud Voice with their telephony system. If that's not you, see the Service Cloud Voice Implementation Guide or Salesforce Help. To update your solution to include Bring Your Own Channel for CCaaS Messaging capabilities along with Service Cloud Voice, see the Bring Your Own Channel Developer Guide.

Omni-Channel Doesn't Show AgentWork to Accept or Reject Incoming Call

Issues you could see:

- Omni-Channel widget doesn't show the AgentWork user interface (UI) to accept or reject the call.
- Omni-Channel user interface (UI) goes blank on incoming call.

Possible causes:

- AgentWork creation failed. Check your connector CALL_STARTED event payload and verify that you don't have invalid data. Also, check that the rep is online with the correct Omni-Channel status for the assigned phone channel.
- Salesforce Omni-Channel has an outage, which prevents pushing AgentWork from the server to the rep console UI. Check Omni-Channel service status at trust.salesforce.com. To verify that AgentWork has been created successfully, refresh the page. Omni-Channel should load the AgentWork and allow the rep to accept or decline the call.

The Connector Doesn't Load Successfully

Issues you could see:

- Omni-Channel widget doesn't show the phone control panel.
- Log in failed in Omni-Channel widget.

Possible causes:

- Your org doesn't have the proper licenses to allow you to use the Service Cloud Voice Partner Telephony product.
- The rep isn't assigned to the required permission set.
- The rep isn't added to the Contact Center.

- The adapter URL isn't configured correctly in your Contact Center.
  - To find out the adapter URL of your Contact Center, you could use any Salesforce public API to read the CallCenter object.
  - The adapter URL in the Contact Center gets inserted from the value in the ConversationVendorInfo record. When you import the Contact Center XML file in Setup, it points to the developer name of the ConversationVendorInfo record.
- The adapter URL isn't accessible.
- The microphone isn't allowed and enabled in the browser.
- CORS-related issue. The connector is loaded in an iframe. You can try to set up CORS for your host in Salesforce Setup.

Partner Telephony Setup Node Doesn't Display

Issues you could see:

- You can't find the Partner Telephony Setup node in Salesforce Setup.
- You can't find the Partner Telephony Contact Center List View node in Salesforce Setup.

Possible causes:

- Your org doesn't have the proper licenses to allow you to use the Service Cloud Voice for Partner Telephony product.
- Service Cloud Voice for Partner Telephony product isn't turned on.
  - If Voice for Partner Telephony isn't enabled in the org, the Partner Telephony Contact Center node that shows the Contact Center list view doesn't appear in the Salesforce Setup.

Phone Control Fails

Issues you could see:

- Some of the phone controls aren't working (for example, accept a call, hold a call, transfer a call).
- Seeing error message or warnings in the Service Console.

Possible causes:

- The connector API isn't implemented properly.

Troubleshooting tips:

- Open the browser developer console and look for errors. Pay attention to the log messages that start with `[sdk]` or `[connector]`.
- Enable the Debug mode in Salesforce to see non-minified JavaScript code in the browser developer console.
- Use the remote simulator to compare the API parameters and responses between the demo connector and your connector implementation.

Download Connector Logs

Partners can log messages from their connector and then download these logged messages as a text file.

See Download Connector Logs.