Service Cloud Voice for Partner Telephony Developer Guide

Version 60.0, Spring '24

'24

CONTENTS

Contents

Contents

Contents

CHAPTER 1   Service Cloud Voice for Partner Telephony Developer Guide

Connect your telephony system with Service Cloud Voice, creating a unified and intuitive agent experience to give customers faster and more personalized service. Salesforce process automation offers recommendations, initiates workflows, and reduces post-call handle time to help agents 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.

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.

Pathways Into Service Cloud Voice

Discover

What is Service Cloud Voice?

📖 Intro to Service Cloud Voice

🔺 Service Cloud Voice Trailhead Module

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 Winter '23

- Added the ability to Set the Voice Call Record Type on page 63 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 76 generated from intelligence sources to the agent console.

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 can be useful when moving to enhanced domains. See Service Cloud Voice for Partner Telephony Apex Reference.
- Added steps on how to Configure Estimated Wait Times for Queues on page 101 and associated fields to the CallInfo, CapabilitiesResult on page 148, and Contact on page 154 objects.
- Added steps on how to Enable the Voice Extension Page in Lightning App Builder on page 92. Also introduced a VoiceExtension metadata type to support the Voice Extension FlexiPage.

3

- 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 Agents. 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 102
- Enable Voice Call Transfers via Salesforce Omni-Channel Flows on page 103
- Connector & Supervisor API
  - CapabilitiesResult on page 148—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 on page 169
  - SupervisedCallInfo on page 170
  - hasTransferToOmniFlow on page 103

Changes to existing topics:

- Quick Start Partner Telephony Package on page 15. 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 79—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 agent work.
- Connector & Supervisor API

- AgentConfigResult on page 141—Moved some of the params to CapabilitiesResult.
- getPhoneContacts on page 131—Added a mention to contact types (not just contacts).
- PhoneContactsResults on page 164—Added a parameter for contact types.
- publishEvent—Added new events for Supervisor Barge In.
- SupervisorEvents on page 173—Added new events for Supervisor Barge In.

Updates for Spring '22

New topics:

- Add Contact Center Channels to Enable Routing on page 90
- Add a Partner Settings UI to Omni-Channel on page 45
- Two-Way Agent Status Syncing on page 93
- Automated Key Provisioning on page 45
- Supervisor Listen In on page 79
- Outbound Dialers on page 61
- Line-Specific Controls on page 117
- Supervisor API on page 168
- Universal Call Recording Playback in Call Recordings on page 83

Changes to existing topics:

- Apex Reference on page 185—Added KeyProvider and PhoneNumberProvider.
- ConversationVendorInfo on page 176—Added keyProvisioningSupported, partnerPhoneNumberSupported, and universalCallRecordingAccessSupported.
- Connector API

  - publishEvent on page 124—Added supervisor events.
  - AgentStatusInfo on page 143—New class.
  - CallInfo on page 144—New class.

- Troubleshooting on page 190—Added a Download Connector Logs section.

CHAPTER 3 Set Up and Publish Your Partner Telephony Package

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.

> **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.

6

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.

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.
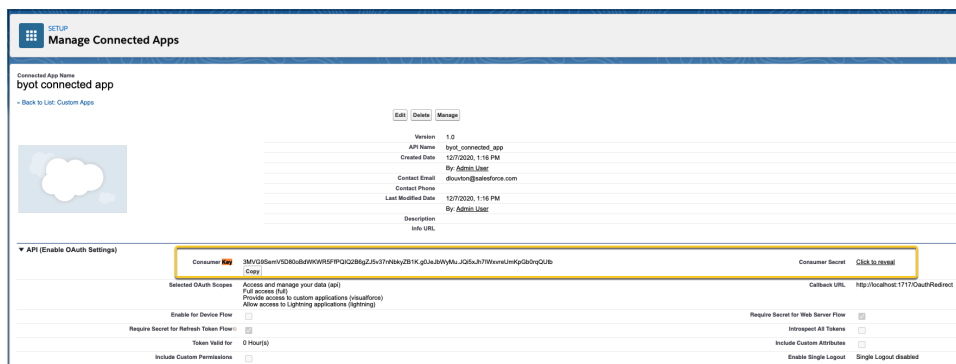
> 💡 Tip: To view the OAuth client ID and personal connected app secret, from Setup, enter `App Manager` in the Quick Find box and click **App Manager**. Click **Connected App Consumer Key** and **Consumer Secret**. To learn about more authentication methods, see Authorize an Org Using the Web Server 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 demo implementation, see this open-source SCV demo connector Github repo.

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-scv-connector, you can use `{your_namespace}:lwcBridge` or `{your_namespace}:AuraBridge`. To learn more see Use the Lightning Message Service Bridge on page 50.
einsteinConversationInsightsSupported	Set this capability to `true` to support Einstein Conversation Insights for customers. Implement the `service_cloud_voice.RecordingMediaProvider` on page 185 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`.
namedCredentialSupported	Set this capability to `true` if you support prescriptive setup through a named credential. Implement the `service_cloud_voice.PartnerConnector` on page 185 interface in your Apex integration class, which tests the

Field	Description
	connection to the new customer account using the customer named credential
partnerTransferDestinationsSupported	Set this capability to `true` so that Salesforce can fetch agent queues in order for Salesforce and agent queues to be mapped. Implement the `service_cloud_voice.TransferDestinationProvider` on page 185 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 Agent SSO using Salesforce as the identity provider. Implement the `service_cloud_voice.PartnerSSO` on page 185 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-scv-connector on page 30, use the absolute link from Use the Demo Connector on page 30. 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 114. 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.
customLoginUrl	The URL that is used to load the telephony system login page. If you aren't using it, remove it. If you're testing with the demo-scv-connector on page 30, use the link from Use the Demo Connector on page 30. 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

Field	Description
	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 185 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 50. 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 185 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: Service Cloud Voice Partner Telephony Sample Application

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`

14

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 on page 176 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	Turns on My Domain in the org
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 19).

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 ...

- **Set Up Omni-Channel and a Lightning Console App**
- **Set Up Service Cloud Voice for Partner Telephony in Your Org**

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.

Set Up Omni-Channel and a Lightning Console App

Service Cloud Voice uses Omni-Channel to send calls to agents in your Lightning console app. Agents 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 get started, assign the automatically generated custom permission, which contains the service presence status set to agents. Then create a Lightning console app and add the Omni-Channel utility to the app.

1. Assign Partner Telephony Permission Set to agents.

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 agents 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.
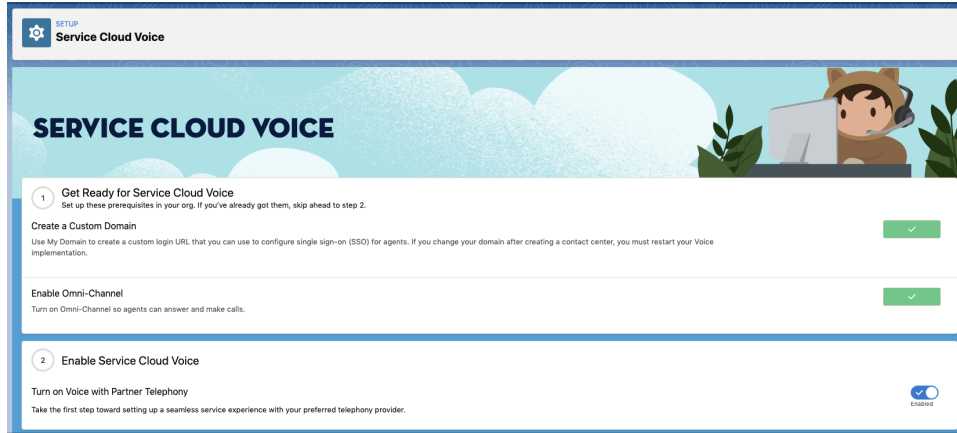
Customers can install the managed package from the App Exchange or from an external URL provided by the telephony partner. Telephony partners can skip installing the managed package from the App Exchange, because they would install the managed package in their scratch orgs in Create an SFDX Project on page 8. Telephony partners can also follow these steps to set up contact centers in their scratch orgs.

After the admin completes the initial steps of creating a custom domain and 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.
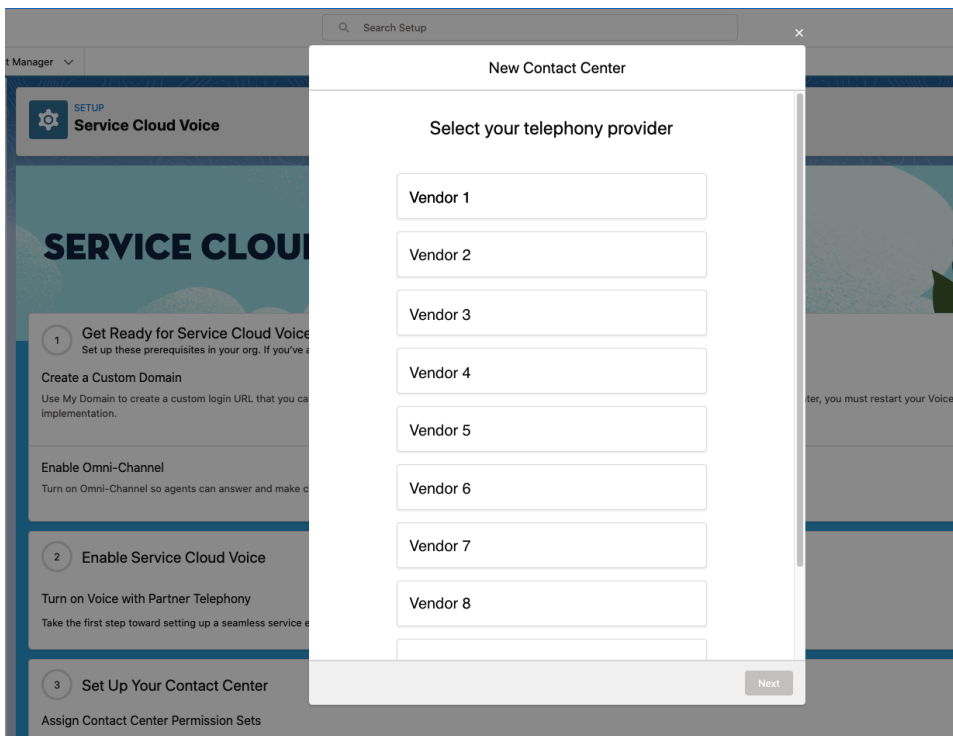
2. After selecting the available vendor from the installed packages, there are two options. Either enter the contact center display name and internal name, or import an XML file to create a contact center.

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 on page 176 setup object. If you don't have a custom object specified in ConversationVendorInfo on page 176, this step is skipped.

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 on page 176 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 agents 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 on page 176 object `namedCredentialSupported` field. If the partner doesn't support this feature, this step is skipped.
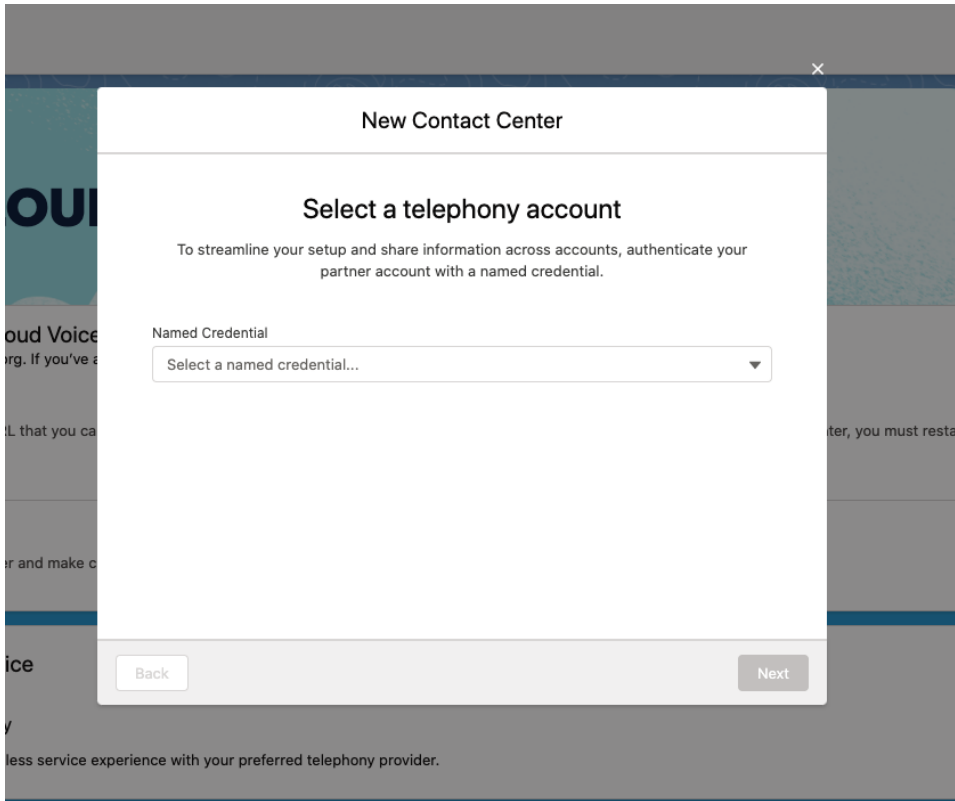
This functionality can be used for Einstein Conversation Insights, user syncing, or retrieving the external queue ID for queue mapping. These features require named credentials for the server-to-server authentication. In this step, the screen allows customers to choose which named credentials to use. 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 185 interface. If the response is successful, it stores the named credential ID in the contact center.

a. If the partner also supports the agent SSO authentication, it must be declared in the ConversationVendorInfo on page 176 object `agentSSOSupported` field. You must also provide an Apex class, which implements the `service_cloud_voice.PartnerSSO` on page 185 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 on page 176 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 185 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.
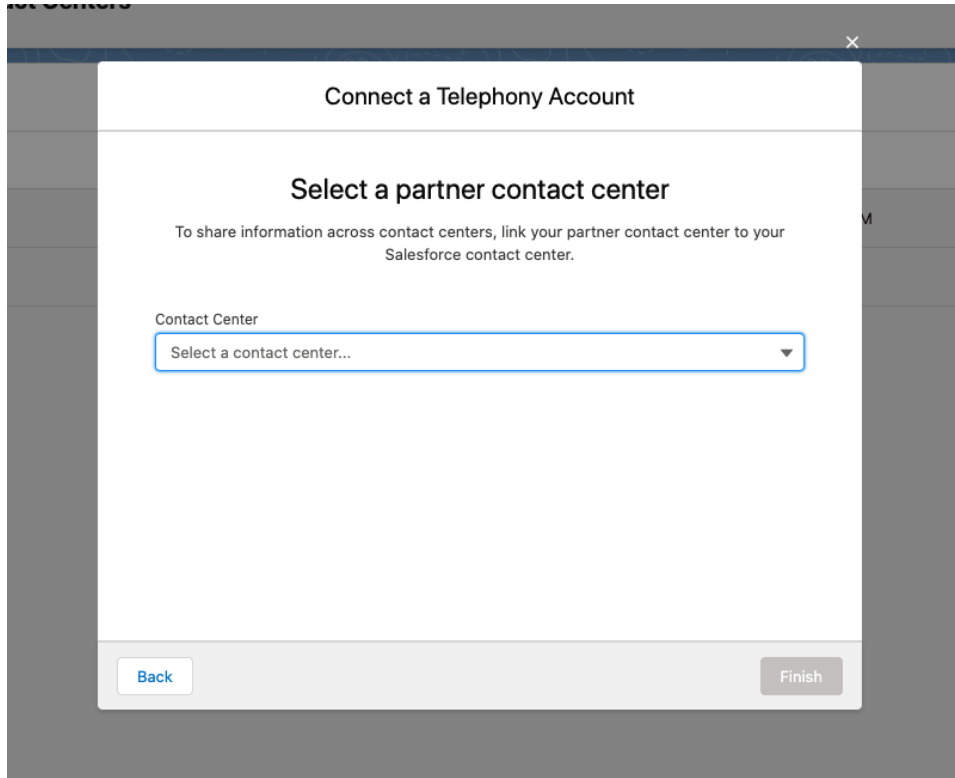
5. When importing XML, the content contains two mandatory sections and some optional, vendor-specific sections.

- 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.
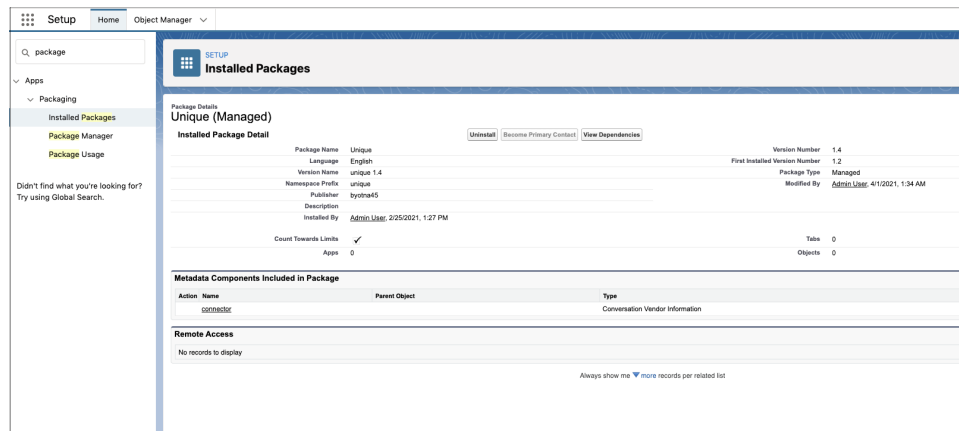
- SCV Settings Section:

- Set the `reqTelephonyIntegrationCertificate` to the certificate as generated in Generate a Self-Signed Certificate with OpenSSL on page 14. 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.

```xml
<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.

Other details about the setup process:

- The admin clicks **Create Contact Center** to launch the wizard.
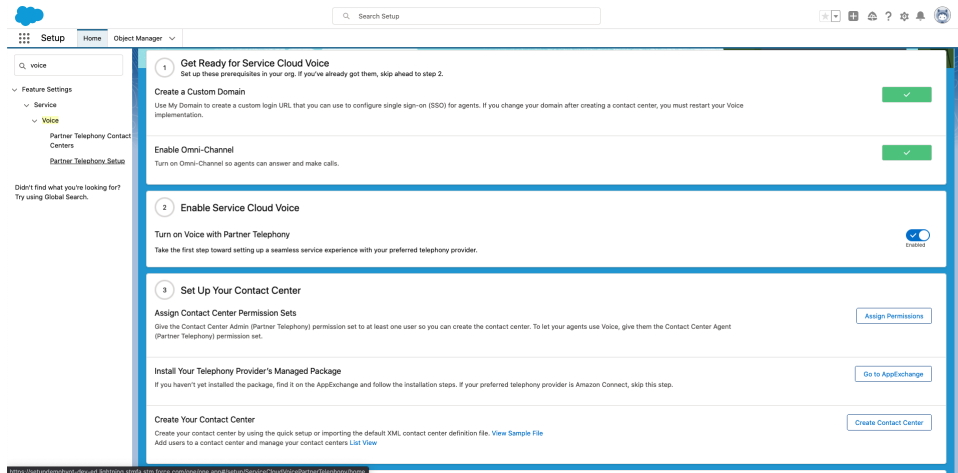
- You can change the values on the contact center details page after the contact center is created.
- If the setup flow is canceled before completion, the steps can still be completed on the contact center details page. The custom settings can be edited in the second section of the contact center details 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
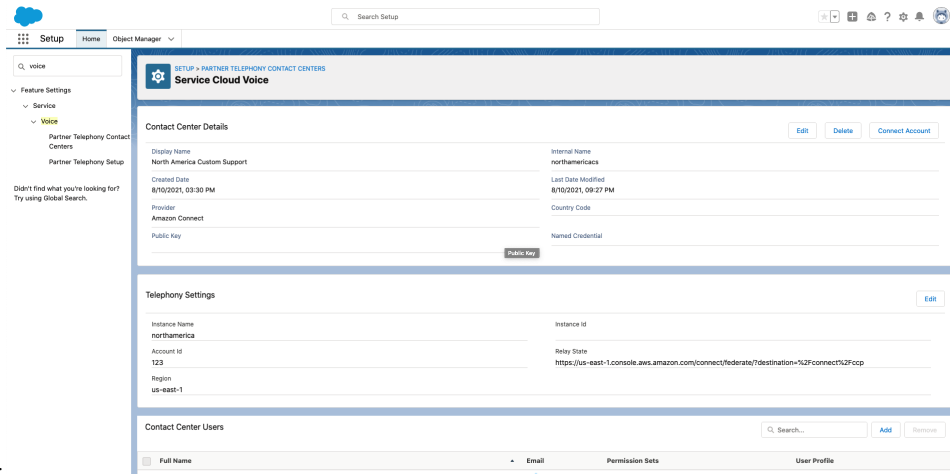
  key.
- After this step is completed, the contact center appears in the Partner Telephony Contact Centers node and the admin can add users to the contact center.

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 agent 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 agent or queue, and send the CALL_STARTED event through the connector.

CHAPTER 6   Connect Your Telephony System to Salesforce

In this chapter ...

- **Use the Demo Connector**
- **Use the Connector API**
- **Test Your Implementation with the Voice Call Simulator**

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.

SEE ALSO:

　　Service Cloud Voice for Partner Telephony Connector API Reference

　　Get Started with the Service Cloud Voice 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-scv-connector

The `demo-scv-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/demo-scv-connector.git
$ cd demo-scv-connector
$ npm install
```

2. Add a private key generated from Generate a Self-Signed Certificate with OpenSSL on page 14.

```
{demo-scv-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 `demo-scv-connector` app uses a self-signed certificate, so you must get your web browser to accept a self-signed certificate.

   a. Open the `demo-scv-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 agent status to **Available**.
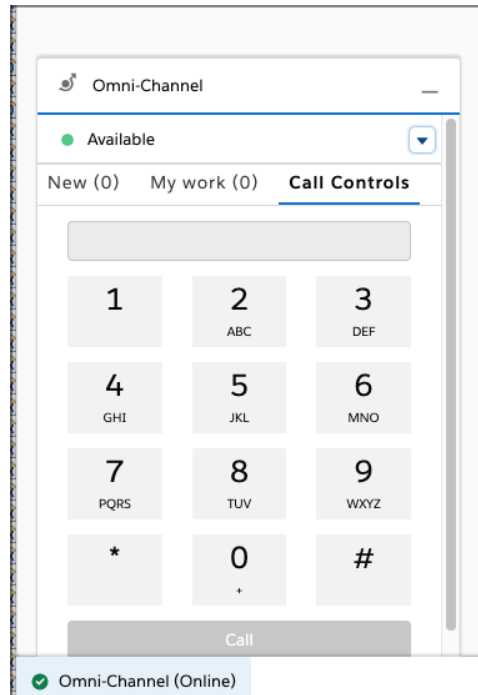
Open the Browser Debugger and make sure that you see messages from the connector. The messages start with `[sdk]` or `[connector]`.

SEE ALSO:

GitHub: Service Cloud Voice Demo Connector

Use the Connector API

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

The Demo Connector on page 30 is a working example that uses the Connector API. For details on using the Connector API, you can review our reference documentation on page 119.

The Connector API on page 119 contains two parts:

1. Connector API Base on page 122: The Connector API Base allows you to initialize the API, and to send information to Salesforce when events occur with your telephony system.

2. Connector API Interface on page 126: The Connector API Interface is what you implement in order to be notified when events occur within the Salesforce org.

SEE ALSO:

Service Cloud Voice for Partner Telephony Connector API Reference

Get Started with the Service Cloud Voice 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 30. 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

| Enter a phone number... | Inbound Call | Outbound Call | Add Participant | Callback |

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)

> Voice Call vendorCallKey

Customer Phone Number (Not Required for Inbound Calls)

> Customer Phone Number

Speaker

| Customer ▼ | Write a message... | Add to Transcript |

[●]

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:
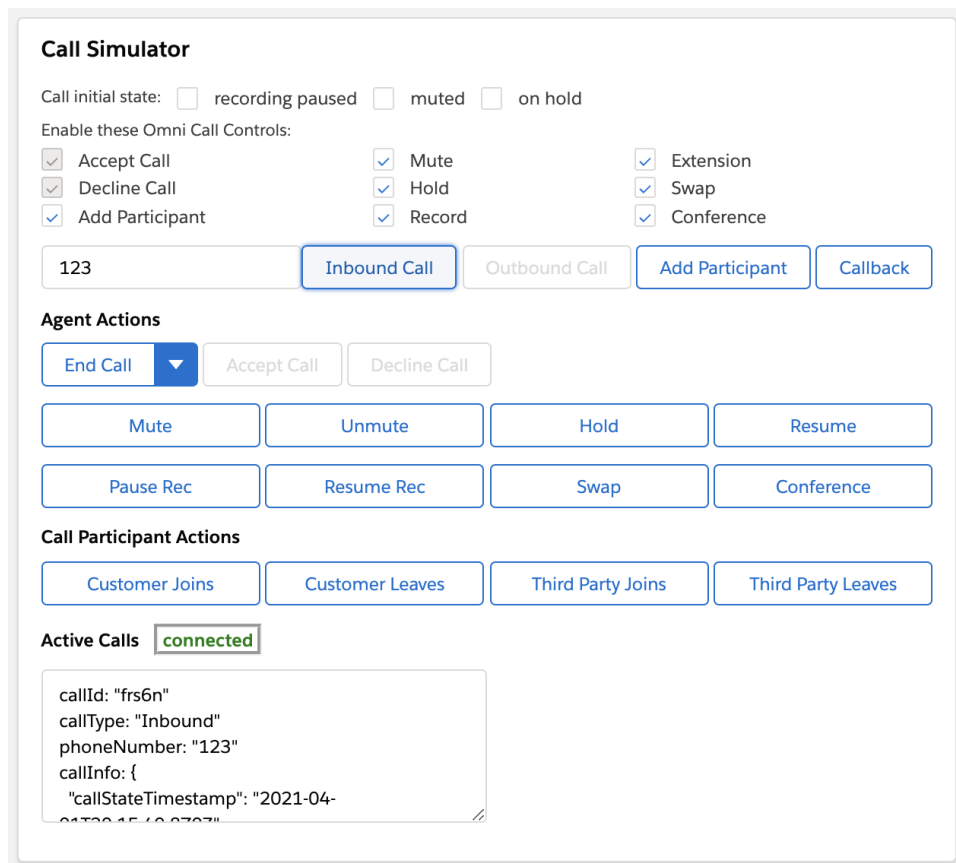
Call Recording

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

Publish

Messages

Communicate with a Lightning bridge component in your org.

Send Message

Use the message format provided in the Partner Playbook.

Send

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.

Log out

- **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 agent 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 50.

- **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**.

- **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.
• **Set Up Single Sign-On**	ⓘ **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.
• **Develop a Telephony System Login Page**	

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. Make sure you've configured a custom domain name. For help, see My Domain.

   a. From Setup, go to **My Domain**. Enter a subdomain name and check its availability.

   b. If the name is available, click **Register Domain**.

   c. Test the domain and deploy to users.

2. From Setup, enter `Identity Provider` in the Quick Find box, then select **Identity Provider** and enable it. Download the certificate and metadata file.

3. Enter `App Manager` in the Quick Find box. Click **App Manager** > **New connected App**.

4. 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.

5. After enabling Identify Provider, download the metadata file for using Salesforce as IdP. This metadata XML file should be used for setting up SSO.
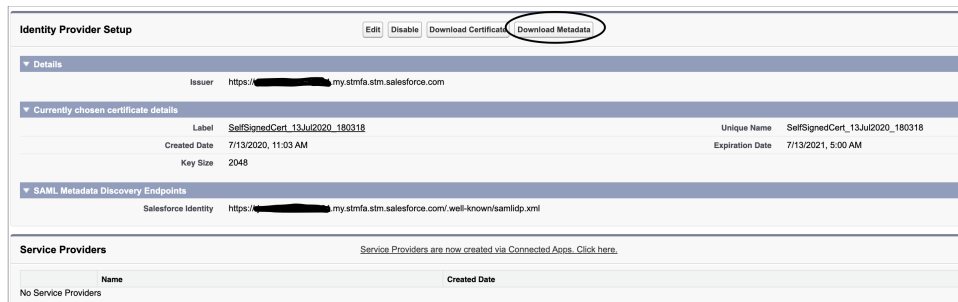
Perform Headless SSO

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` on page 158 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 38

- [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 agent to make outbound calls.

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

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 agents 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">
...
```

ctristypertraptDaperTeerappsSKingssekjhpapns/SPKYRerappsptMapnts/ciriifnaparefridapeTppapeiRsprapppnsSrilikpcpDpaps46sgrilysperTinpptsens

```
...
</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:
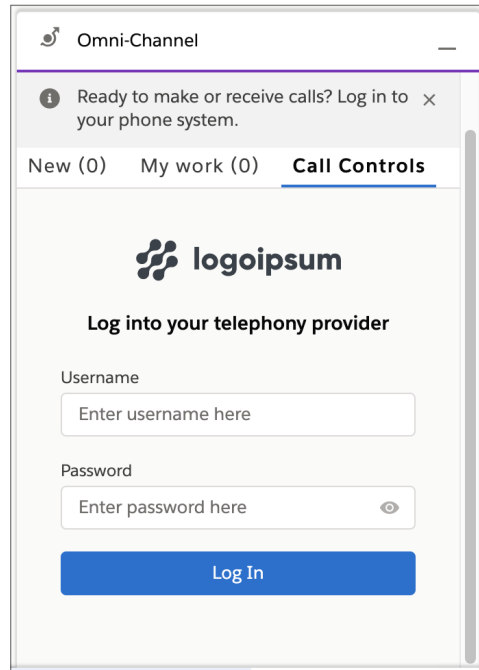
4. Publish a Connector API event on page 124 with the login result to show either a login failure or the dialer.

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

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

42
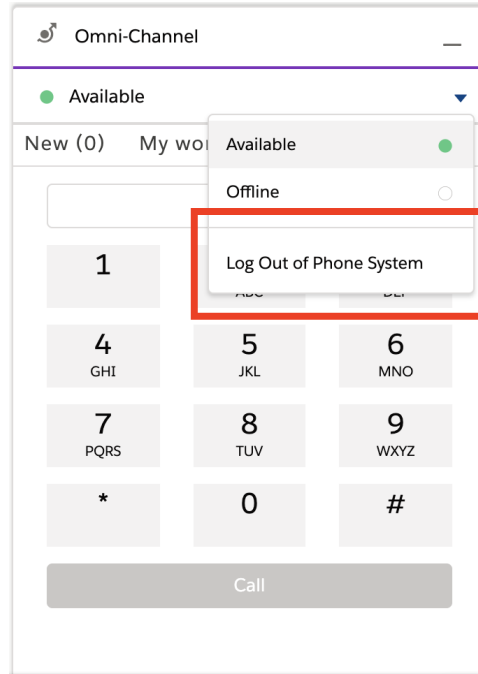
6. In the event of a session timeout, the telephony vendor can trigger a logout on page 134 from the Connector API and use the Connector API publishEvent on page 124 method. The vendor can provide a loginFrameHeight for the login iframe. If no value is provided, a default value of 350 is used.

```
publishEvent({
  eventType: Constants.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 32.

SEE ALSO:

logout

CHAPTER 8   Customize Your Implementation

In this chapter ...

- **Automated Key Provisioning with Service Cloud Voice for Partner Telephony**
- **Add a Partner Settings UI to Omni-Channel**
- **Customize Error Messages**
- **Communicating with Lightning Components**

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.

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 centers list page.
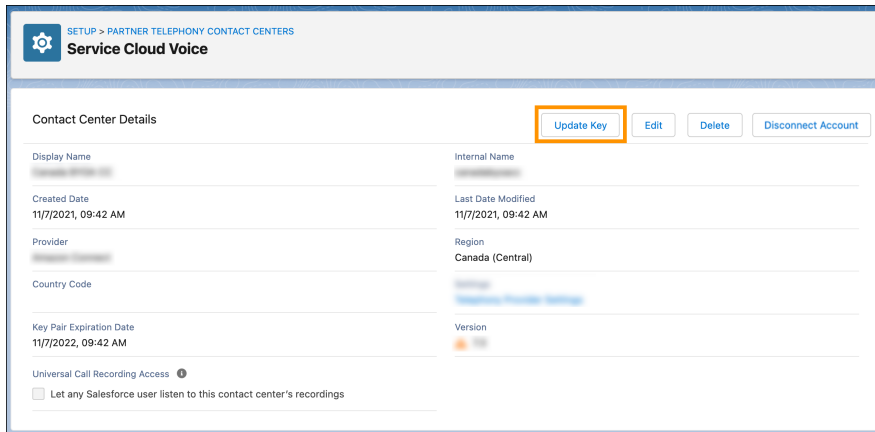
For automated key provisioning and renewal, the following requirements must be met:

1. Partners must set the `keyProvisioningSupported` capability to `true` in their ConversationVendorInfo on page 176 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 a public key that is expired, 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 agent settings.
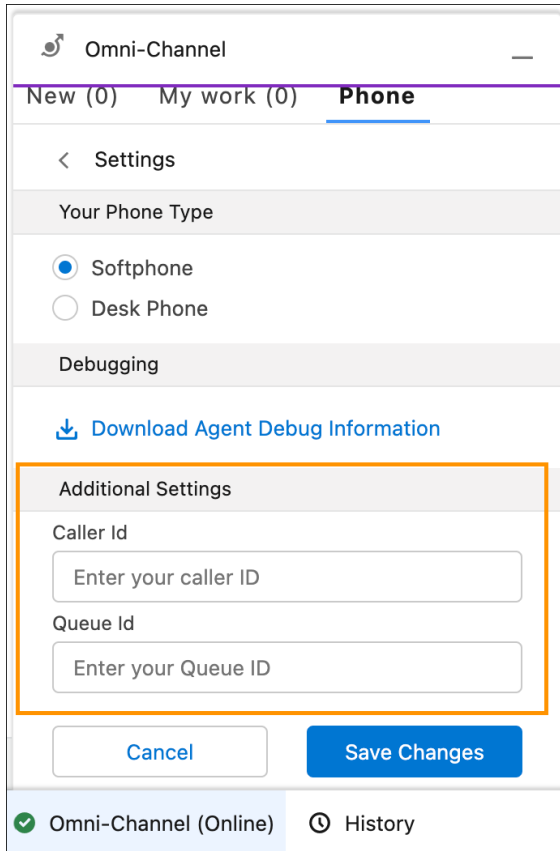
This Lightning web component (LWC) can show any available LWC elements, such as radio boxes, checkboxes, comboboxes, input boxes.

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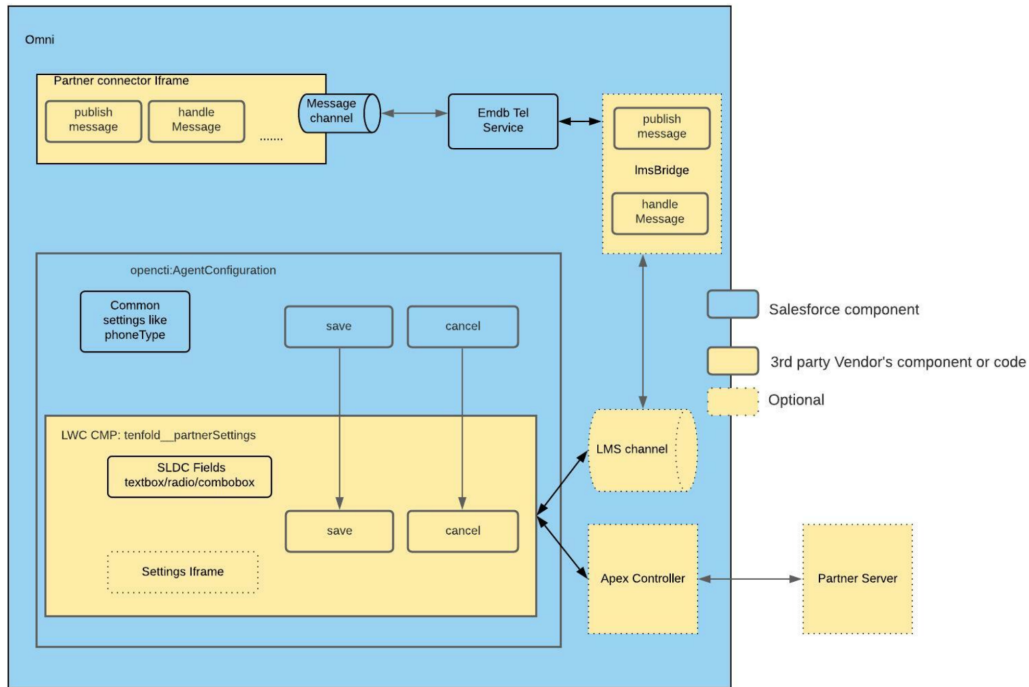
In Conversation Vendor Info, set the `TelephonySettingsComponent` field to {namespace}:{componentName}.

Sample `agentConfig.html`:

```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`:

```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 agent 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 agent 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 agent 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 Service Cloud Voice 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 Service Cloud Voice 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

Connector API: publishError (Connector API Base)

Connector API: CustomError (Connector API Object)

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 on page 176 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 on page 176 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 the CallCenter 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">
...

...
</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 32.

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.

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 32.

The call appears in the Omni-Channel utility.
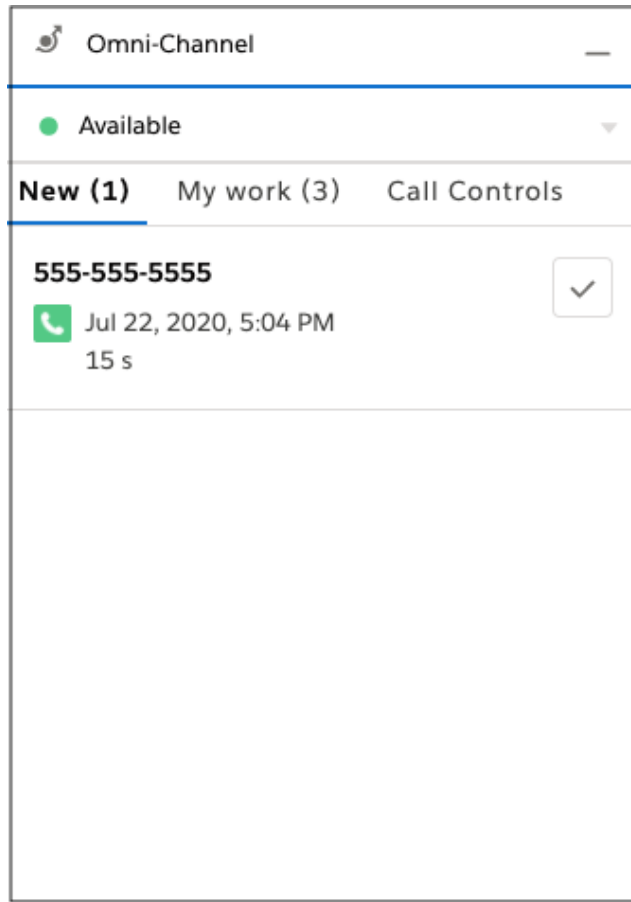
To accept the call, click the checkmark. You can mute, unmute, transfer, and end the call. Salesforce opens the VoiceCall record that is created.
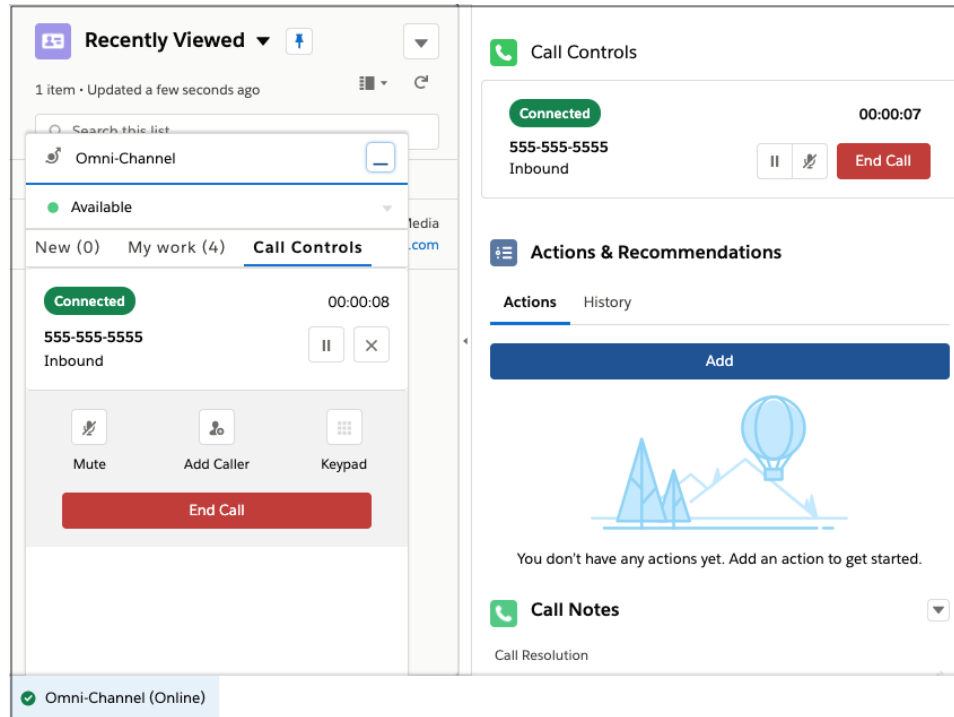
Integrate Incoming Voice Call Creation

For inbound calls, when the telephony system receives the incoming call and prepares to route the call to an agent, 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 38.

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 14. 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

Make Outbound Calls in Omni-Channel

Outbound calls can be made in two ways. Agents can dial a phone number in the Omni-Channel utility Phone tab, or they can click a phone field to initiate an outbound call from a contact, case, or other Salesforce record.

1. Open the Omni-Channel utility and dial a number.
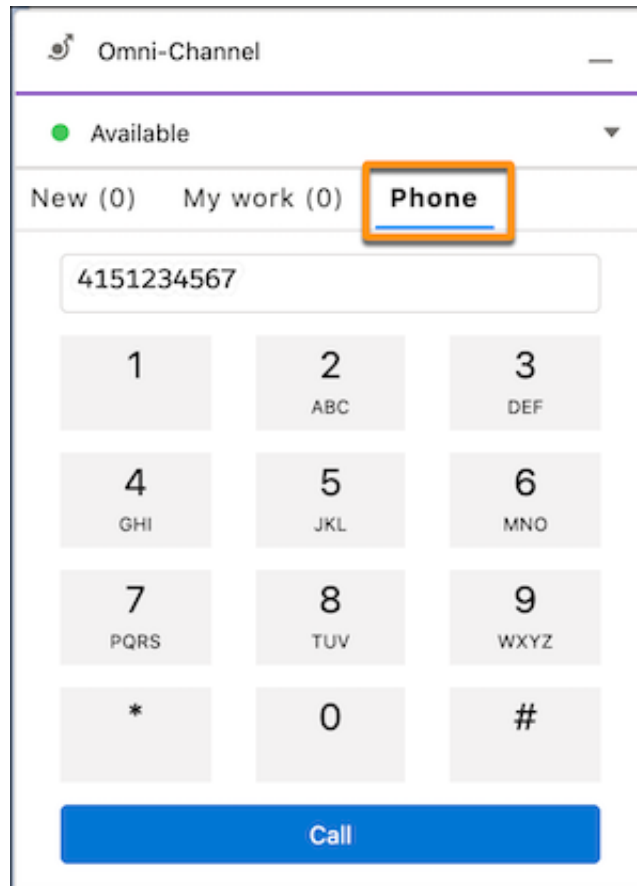
2. Verify that the number is called. The phone call appears in the Omni-Channel utility, first as "Dialing". It can be changed to "Connected" when **Customer Joins** is clicked on the remote controller, which in turn dispatches the event CALL_CONNECTED.
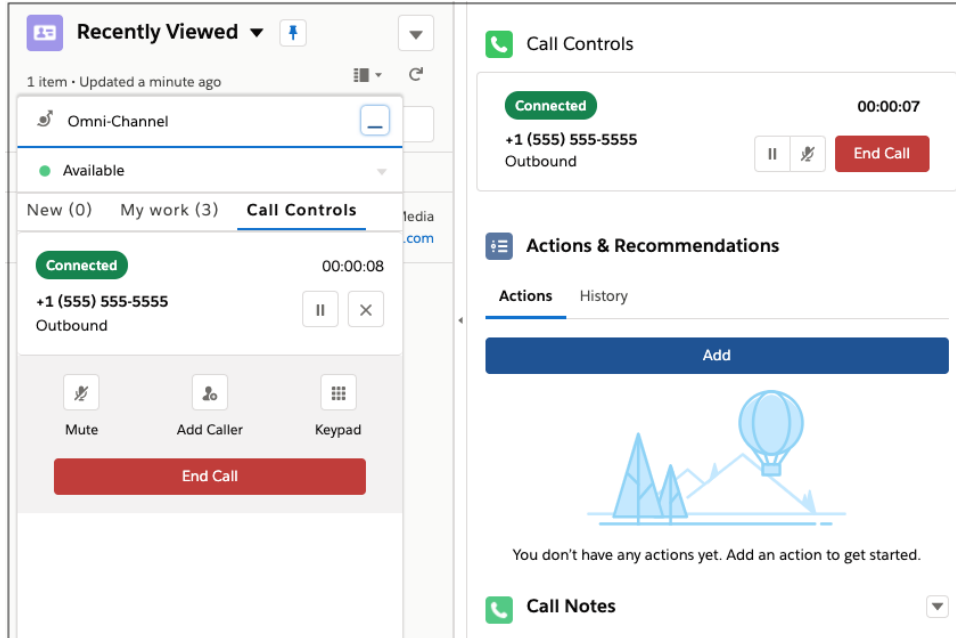
SEE ALSO:

Service Cloud Voice for Partner Telephony Connector API Reference

Get Started with the Service Cloud Voice Connector API

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 agents 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 an agent 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 agent 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 agent 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 agent 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 agent. If no agent is available and the customer requests a callback, you must publish a QUEUED_CALLBACK_STARTED event.

Call the publishEvent() on page 124 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 agent work 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.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 agents 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 an agent is available) or a callback is scheduled for a customer on a particular date.

A callback can also be requested from a particular agent. The telephony system creates a new voice call VC2 and the connector sends a QUEUED_CALL_STARTED event as per agent 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 an agent 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.
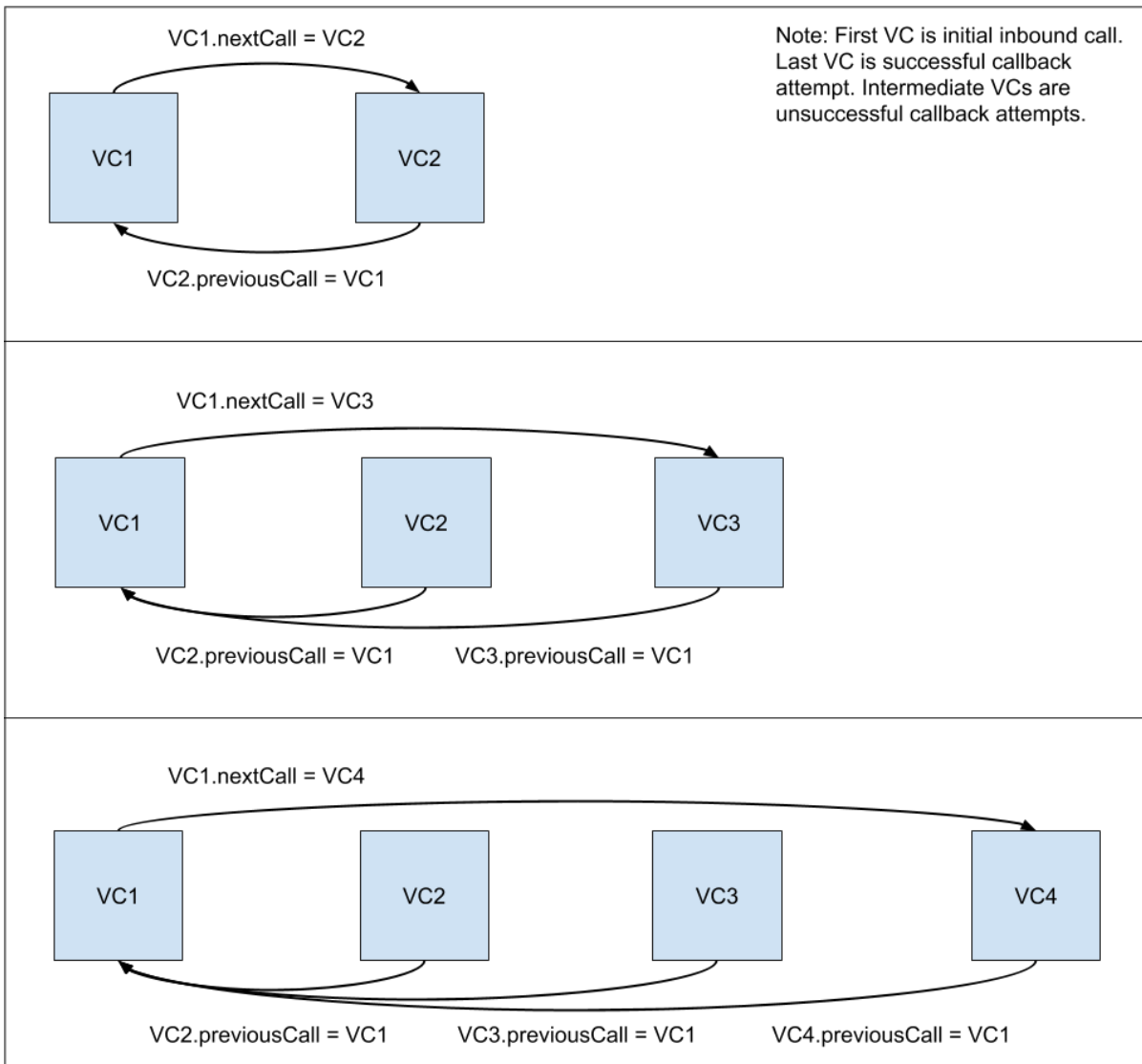
Callback acceptance and outbound call dialing:
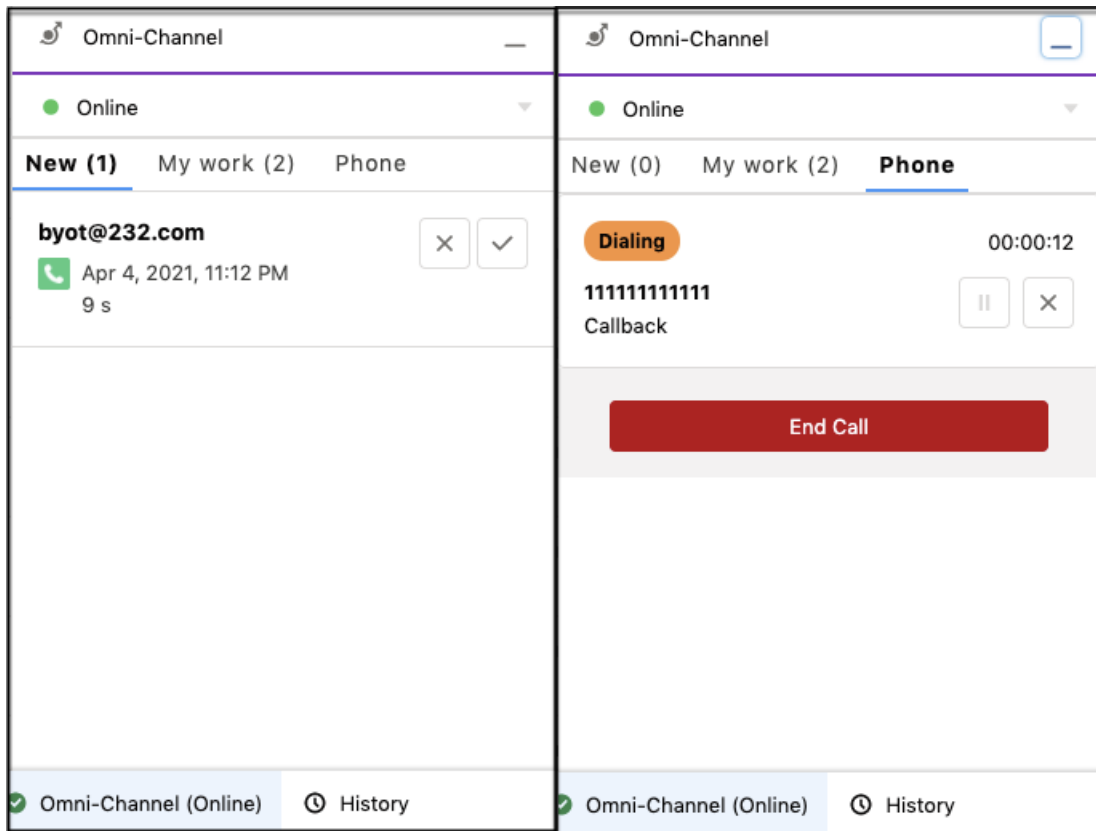
If an agent declines the call or the call times out, the telephony system can try to connect to another available agent with the same VC2.

SEE ALSO:

publishEvent

CallResult

PhoneCall

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

This feature leverages the getPhoneContacts() on page 131 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() on page 131, set the value of `hasPhoneBook in CapabilitiesResult` to `true`.

2. When implementing `dial()` on page 130, set the appropriate `callType` for the agent and queue contact types. For example, if contact type is 'agent', set `callType` to InternalCall, which represents a call made between agents within a contact center. In our demo implementation of `dial()` on page 148, 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 agent-to-agent calls at the org level by selecting the **Voice Call (VoiceCall)** Change Data Capture entity.

4. When an agent makes a phone book InternalCall, the agent's `participantType` should be `Initial_Caller`, and the receiving agent's `participantType` should be `Agent` in your `CallResult`. See the demo implementation of `dial()` on page 130 for an example.

5. To ensure the agents' 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 agents share the same voice call record for `callType InternalCall`, the first agent (Agent 1) is a standard user with restricted access to the VoiceCall object, while the second agent (Agent 2) becomes the owner of the VoiceCall record. This might cause Agent 1 to lose access to the VoiceCall record. To ensure Agent 1 can access the VoiceCall record, create sharing rules based on group membership by adding all agents 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 agent and queue contacts appears.

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

SEE ALSO:

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, agents 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 Agents

Let customers send your agents voicemails that agents 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 an agent, if the transcription is still being processed in Salesforce, it may not show up when the agent accepts the voicemail. If that's the case, the agent 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.

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 agents 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 agent in the Omni-Channel widget. These buttons can be hidden or shown for each call. By default, buttons are visible.
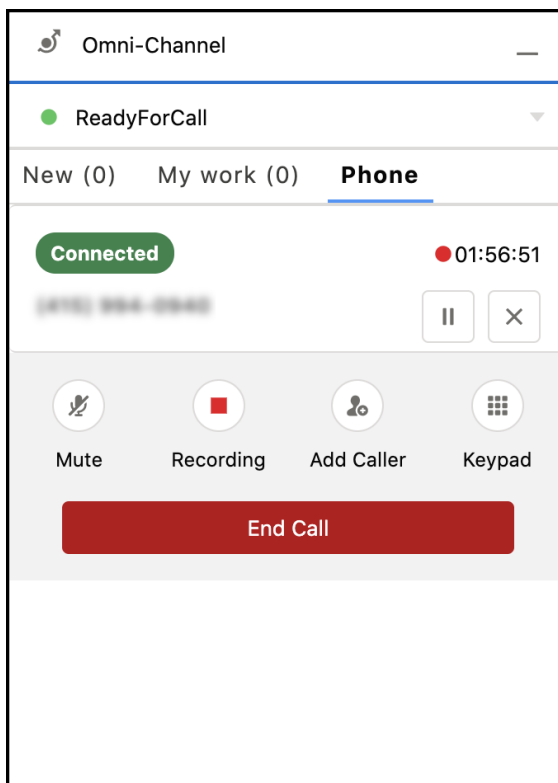
These buttons can be hidden from view.

- 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` on page 144 object. For instance, use `showAddCallerButton` to hide or show the add caller button. The `CallInfo` on page 144 object is used by the `PhoneCall` on page 162 object for methods such as `acceptCall` on page 128.

CHAPTER 10 During Call Actions

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.

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**.
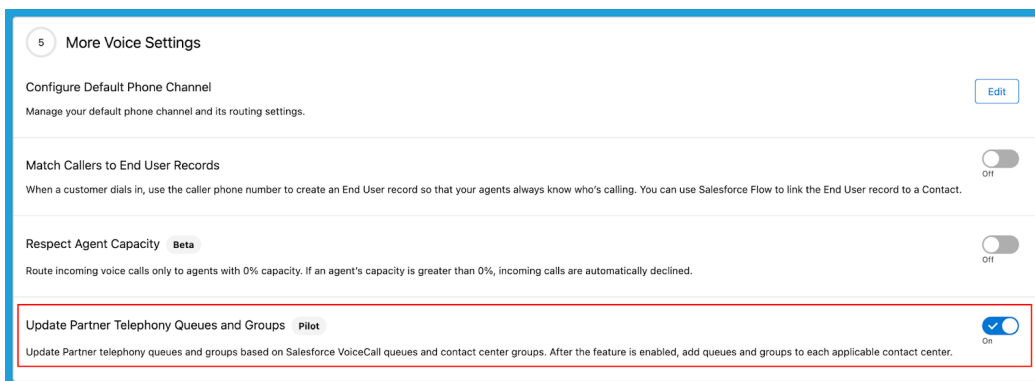
3. When the vendor imports the `ConversationVendorInfo` record, they must set the following fields.

- `CapabilitiesSupportsQueueManagement`. Set this value to `true`.
- `IntegrationClassId`. This value contains the ID of Apex implementation class. For example, `01pxx000000wxyzABC`.

Field	Value
Id	0
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	0
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.
```

71

```
        // 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.
```

74

```
        // 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, agents can change their Omni-Channel status while on a call.

To use this feature, vendors should set the value of hasPendingStatusChange in CapabilitiesResult on page 148 to true. After Salesforce receives a value of true from the getCapabilities() method, the Omni-Channel status change button is enabled for agents during a call.

Then, when setAgentStatus on page 137 requests are made, Salesforce supports an additional parameter, enqueueNextState, and the vendor can implement the enqueue status change feature to support this behavior.

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 agent.
- 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

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 an agent 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` on page 148 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()` on page 131, have `hasSupervisorListenIn` set to `true` in the returned `CapabilitiesResult` on page 141. 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.
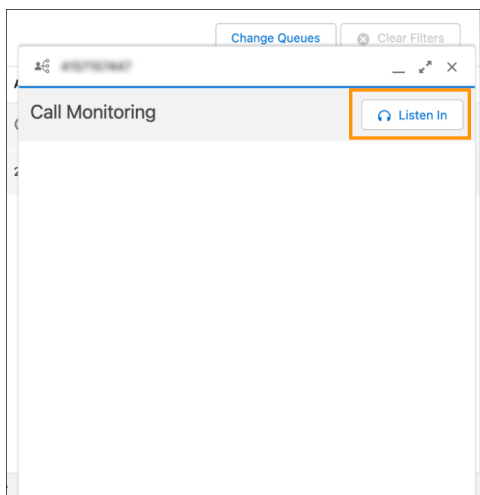
Supervisor Listen In Details

When a supervisor clicks the **Listen In** button, the call information is delegated to the vendor's connector
superviseCall(supervisedCallInfo) on page 169 method. The superviseCall(supervisedCallInfo) on
page 169 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 on page 170.

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) on page 172 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.EVENT_TYPE.SUPERVISOR_CALL_CONNECTED, new
SuperviseCallResult(call) });
  } catch (e) {
    publishError(constants.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 on page 144 and the capability
CapabilitiesResult.hasRecord on page 148 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` on page 148 and
   `CapabilitiesResult.hasSupervisorBargeIn` on page 148 are both true.

4. The call is connected.

When clicked, we call the same `pauseRecording()` on page 134 and `resumeRecording()` on page 135 functions.

CHAPTER 11   Post-Call Actions

In this chapter ...

- **Call Recordings**
- **Post-Call CTR Sync with the Update VoiceCall API**
- **After Conversation Work**
- **Mean Opinion Score (MOS)**

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.

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 86, 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 agent is logged into the telephony system through SSO or a login page. Otherwise, the agent 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 an agent 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.
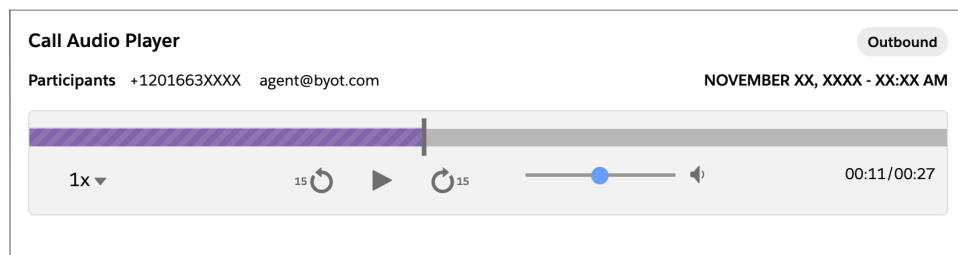
The callDurationInSeconds field on voice calls is determined by calculating the difference between the startTime and endTime fields, and is unrelated to call recording duration.

> 📝 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 demo-scv-connector on page 30 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

Publish

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 getSignedRecordingUrl on page 131 API in their connectors. This API accepts vendorCallKey, callId, recordingUrl as parameters and returns a SignedRecordingUrlResult on page 166 object.

The vendor specifies `hasSignedRecordingUrl` as true. When the recording component loads, Salesforce issues a request to the getSignedRecordingUrl on page 131 API 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 getAgentConfig() {
    return new AgentConfigResult({
        hasSignedRecordingUrl: true
    });
}

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

Pause and Resume Recording

Agents 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 APIs to allow users to pause and resume recording:

- pauseRecording(call) on page 134
- resumeRecording(call) on page 135
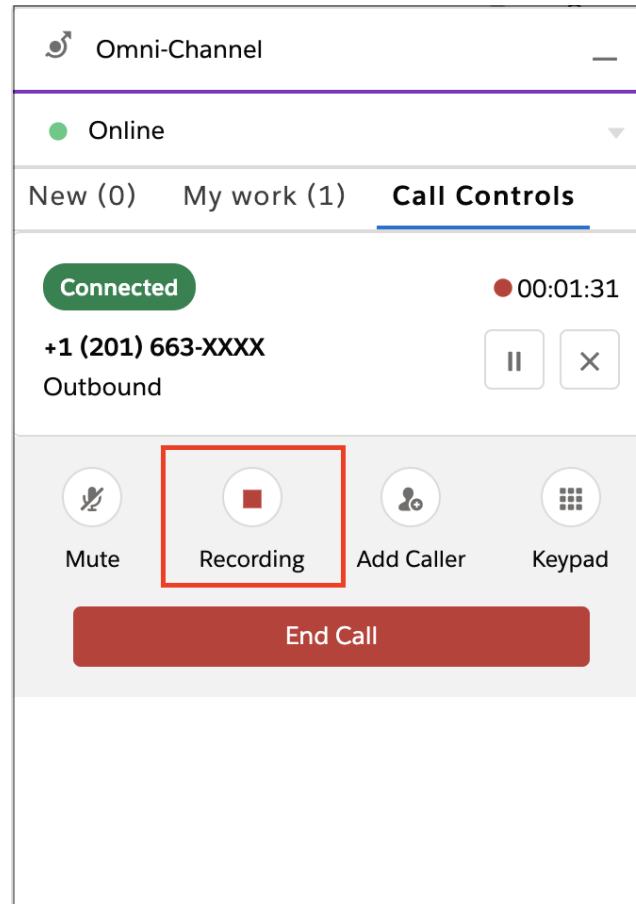
Universal Call Recording Playback

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 on page 176 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**.

When a warning dialog appears, click **Allow**.
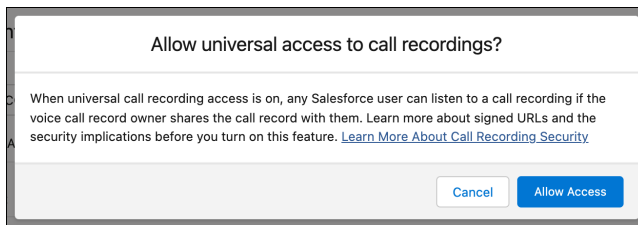
SEE ALSO:

Service Cloud Voice for Partner Telephony Connector API Reference

Get Started with the Service Cloud Voice 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 orgsm 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 38.

This API uses JWT authorization to communicate with Salesforce.

SEE ALSO:

Voice Implementation Guide: Update VoiceCall

After Conversation Work

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

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

1. Fire an AFTER_CALL_WORK_STARTED event (using the `publishEvent()` on page 124 method) with the callId to trigger after conversation work for the agent.

   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 agent 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.EVENT_TYPE.AFTER_CALL_WORK_STARTED,
  payload: { callId: <uniqueCallId> }
});
```

2. Implement the `wrapUpCall()` on page 138 method in the connector. The implementation of this method should remove the agent from wrap-up and put the agent 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 agent wraps up a call, fire WRAP_UP_ENDED (using the `publishEvent()` on page 124 method) event after you receive `wrapUpCall()` on page 138, with the following sample payload.

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

SEE ALSO:

Get Started with the Service Cloud Voice Connector API

wrapUpCall

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 agent configuration the field `config.supportsMos` using the `setAgentConfig` on page 136 method.

2. Publish the UPDATE_AUDIO_STATS event using the `publishEvent` on page 124 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` on page 142 as `true` in the `AudioStats` on page 142 payload of the `publishEvent` on page 124 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` on page 142 as `true` in the last `UPDATE_AUDIO_STATS` `publishEvent` on page 124 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.

CHAPTER 12 Route Calls

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.

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 agent and the agent accepts the Agent Work. If the voice call is never
> routed to the agent or the agent declines the Agent Work, 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**.
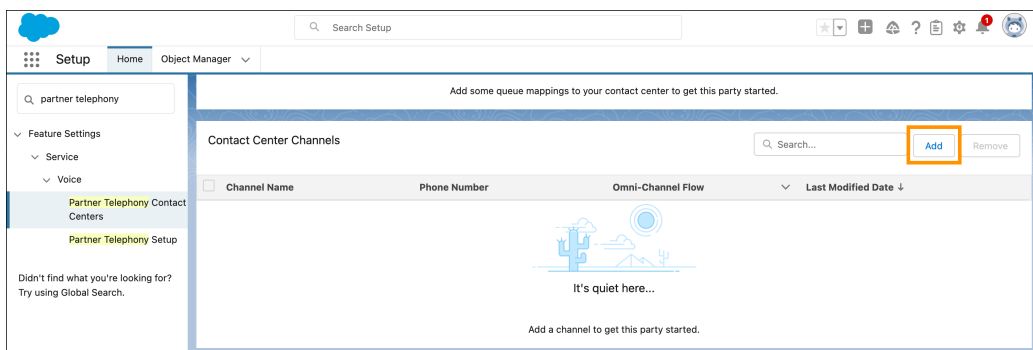
Partners can automate the phone number selection by providing the phone number list through an Apex implementation. Otherwise,
the phone input box displays a text box where the agent can manually enter the phone number.

Partners can turn on the capability `partnerPhoneNumbersSupported` in ConversationVendorInfo on page 176 and implement the `service_cloud_voice.PhoneNumberProvider` interface in their Apex class on page 185 as shown in the following codeblock.

```
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 Agent Mapping

This topic provides guidance on how to handle queue mapping and agent 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 on page 176 record.

2. Implement the Apex interface service_cloud_voice.TransferDestinationProvider on page 185.

Agent Mapping

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

1. Set `userSyncingSupported` to `true` in the partner `ConversationVendorInfo` on page 176 record.

2. Implement the Apex interface service_cloud_voice.UserSyncing on page 185, 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 agent 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 Agent Statuses

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

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

Agent Action in Salesforce	Agent Status in Omni	SetAgentStatus sent to Connector	Connector Event/Callback	Agent 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

Agent Action in Salesforce	Agent Status in Omni	SetAgentStatus sent to Connector	Connector Event/Callback	Agent 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 an agent 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:

Get Started with the Service Cloud Voice Connector API

setAgentStatus

Two-Way Agent Status Syncing

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

For example, a telephony partner system may have scheduled time breaks for the agents where they would want to change the agent status in Omni accordingly. If an agent 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()` on page 133 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()` on page 124 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()` on page 137 for information on how to update the vendor agent status when the Omni-Channel status changes.

SEE ALSO:

Service Cloud Voice for Partner Telephony Connector API Reference

Handling Missed Calls and Call Errors

A voice call typically ends when an agent hangs up. Voice calls can also end when an agent 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 an agent clicks the **End Call** button, invoking the vendor's endCall() on page 130 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 an agent 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 publishEvent() on page 124 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 an agent misses a call, set `call.agentStatus` to `MissedCallAgent`.
- When an agent declines a call, set `call.agentStatus` to `DeclinedByAgent`.
- When a call fails due to the agent, set `call.agentStatus` to `FailedConnectAgent`.
- When a call fails due to any reason that's unrelated to the agent, set `call.agentStatus` to `FailedConnectCustomer`.
- When a customer misses a call, set `call.agentStatus` to `MissedCallStatus`.

To set the agent'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 agent declines a work item.

To set the agent'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:

Get Started with the Service Cloud Voice Connector API

publishEvent

CallResult

PhoneCall

External Routing

Omni-Channel routes work to agents using a two-step process. In the first step, a PendingServiceRouting (PSR) is created which represents a work assignment that's waiting to be routed. If this PSR's RoutingModel field is set to ExternalRouting, Omni-Channel won't route the PSR and waits for the vendor to create the AgentWork record for the assigned agent using the PSR.

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

As an example, here's what a vendor system does for an agent to accept a work item.

1. Chat visitor initiates a chat request from the external routing button.
2. `PendingServiceRouting` is created.
3. Partner is notified by a pushTopic event (EventType = Create, isPushed = false).
4. Partner creates AgentWork using the PSR.
5. Agent is routed to the chat request (AgentWork Status = Assigned).
6. Agent accepts the chat request (AgentWork Status = Accept).
7. Omni-Channel deletes the `PendingServiceRouting` record after the agent accepts the work.

8. Partner is notified by a pushTopic event (EventType = Delete).

SEE ALSO:

Salesforce Help: The Routing Lifecycle

CHAPTER 13 Transfer Calls

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.

Transfer Calls in Omni-Channel

Learn how to transfer calls in Omni-Channel. We recommend using the Voice Call Simulator to test different transfer scenarios.

For details, see Test Your Implementation with the Voice Call Simulator on page 32.

To transfer a call, click **Add Caller** in the Omni-Channel utility. Then, either use the keypad to dial a number directly or use the speed dial to select a contact.

Add caller:
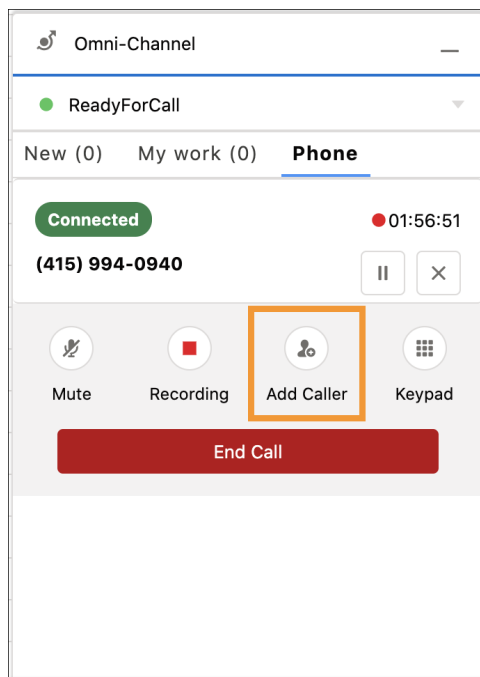
Dial from the contact list:
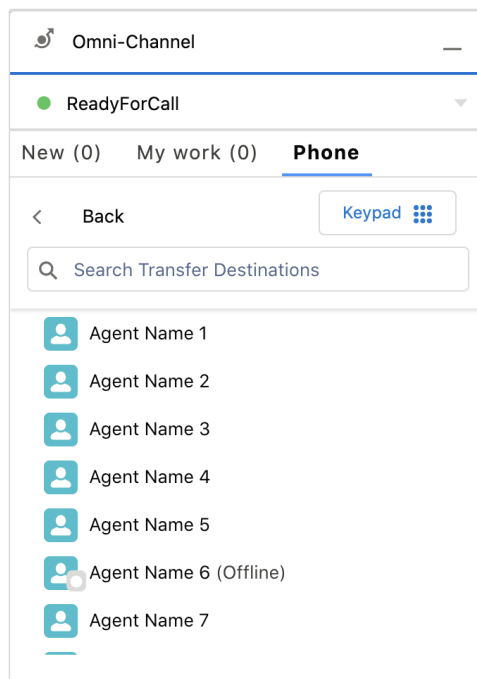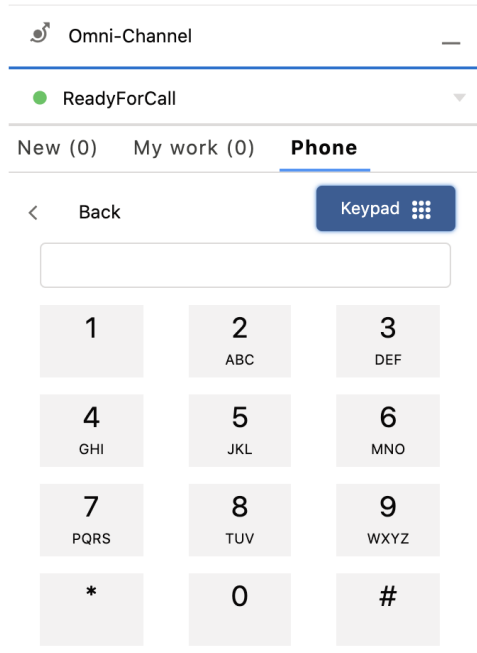
When the call is dialing, pick up the transfer call to connect the call. If you're using the Voice Call Simulator, click **Third Party Joins Call**.
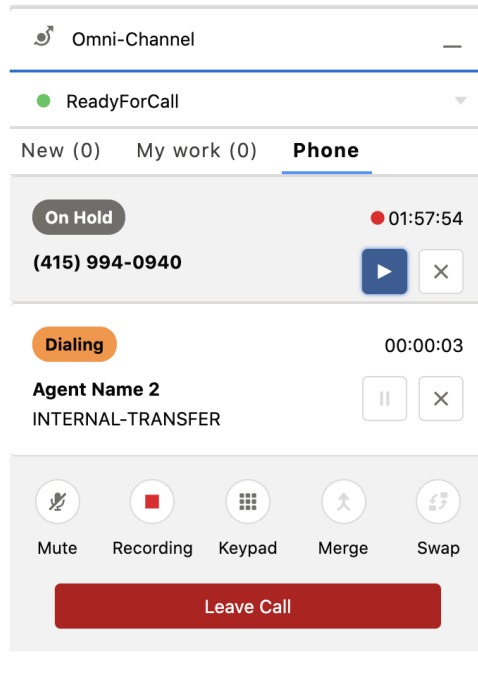
Click **Swap** to switch which participant is on hold.
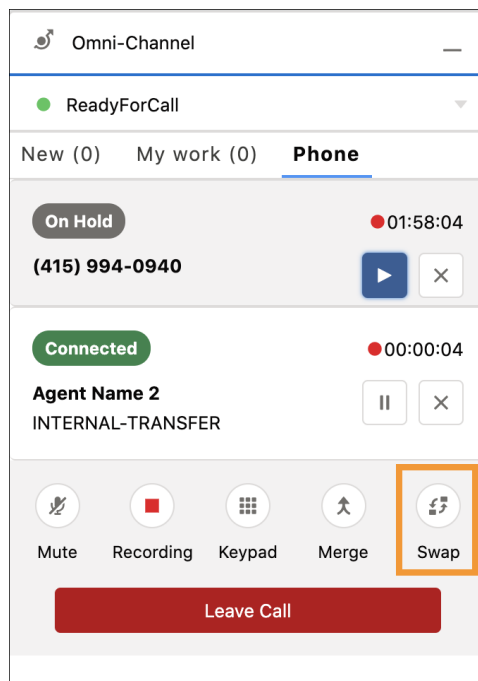
Click **Merge** to merge the participants and agent into one call.
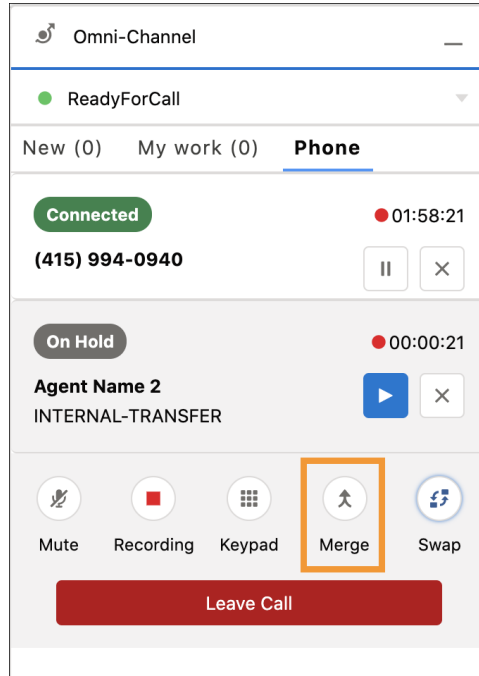
If needed, remove a participant. In the simulator, click **Third Party Leaves Call**. To remove the first participant from the call, click **Customer Leaves Call**.

Configure Estimated Wait Times for Queues

Agents can see estimated wait times for a given queue 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

Salesforce calculates estimated wait times only if 10 calls were made to the queue over 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:

1. Verify the `hasQueueWaitTime` field in the CapabilitiesResult on page 148 object is set to false. This is the default value.

2. Verify the telephony provider queues are mapped to Salesforce queues.

3. When accepting a call, provide the telephony provider's `queueId` value to the CallInfo on page 144 object.

4. 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 on page 154 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:

  Salesforce Help: Map Your Salesforce Queues to Telephony Provider Queues

  Salesforce Help: Make Smarter Routing Decisions by Checking Agent Availability Queues

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 on page 141.

2. **Salesforce-Provided Availability**. If the partner doesn't provide agent availability as part of the phone book contacts (`hasAgentAvailability` is `false` in AgentConfigResult on page 141), 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

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, agents can transfer a caller to any of the following transfer destination types through the Omni-Channel widget:

* Agent

* Contact

* Queue

* Flow
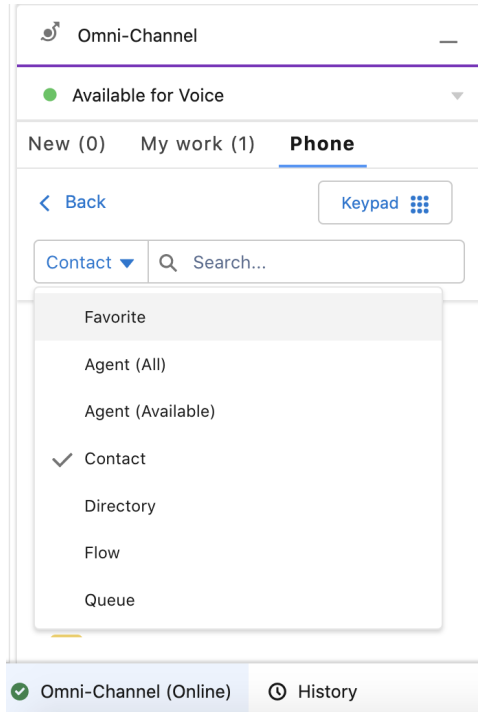
By default, the widget displays all transfer destination types, giving agents the option to transfer a call to a specific agent, contact, queue, or flow.

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

- Using the Connector API, set the value of `contactTypes in PhoneContactsResult` on page 164 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 agents to pick from.

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 agents 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 on page 148 is set to `true`.

When an agent transfers a call to an Omni-Channel flow, Salesforce calls the addParticipant on page 128 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 demo-scv-connector script.

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` on page 148 to `true`. See `getCapabilities()` on page 131.

2. Update the implementation of `addParticipant()` on page 128 to perform a blind transfer when the `isBlindTransfer` argument is true. A sample blind transfer is implemented in the Demo Connector on page 30.

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 on page 124.
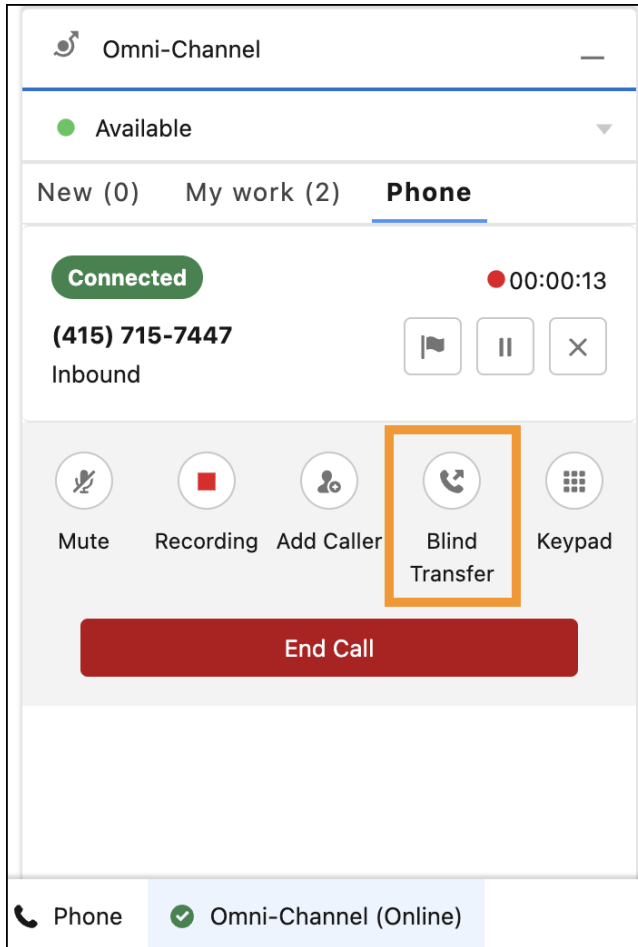
Use Click-to-Dial for Transfers

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

Phone numbers (using the lightning:clickToDial Lightning component) are clickable when an agent isn't on an active call. Clicking the phone number starts an outbound call. An agent 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 agent is prompted with two options. The agent 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 an agent adds a participant to a call, Salesforce provides a UI to search for transfer destinations.
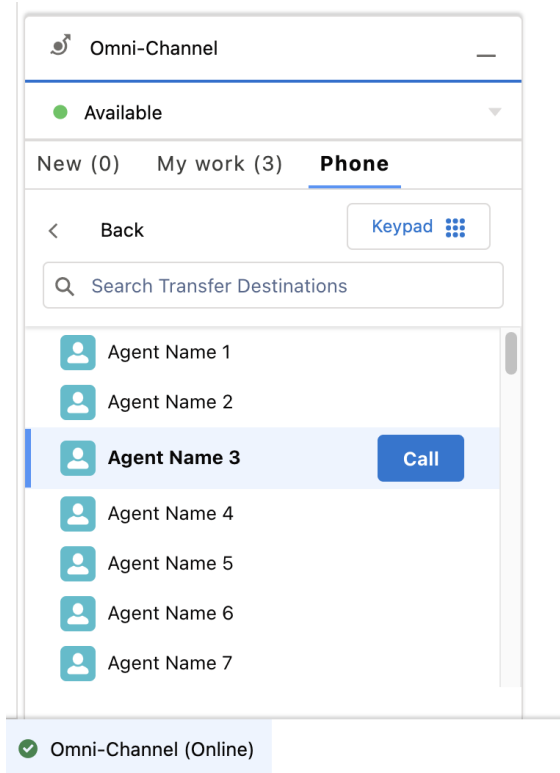
The contacts are returned from the vendor when implementing `getPhoneContacts()` on page 131.

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

Search and Pagination

`getPhoneContacts()` on page 131 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` on page 141 as `true`), the following events occur.

- When the transfer UI is shown, `getPhoneContacts({limit: 50})` on page 131 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 })` on page 131.

- When scrolling down ("progressive loading"), Salesforce calls `getPhoneContacts({contains: string, filter: string, limit: 50, offset: calculated})` on page 131.

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()` on page 131 is called one time.

CHAPTER 14 Desk Phone Support

If the telephony provider supports desk phones, agents can make outbound calls or answer inbound calls from their desk phone. Agents can also decline calls and initiate transfers from their desk phone.

> ⚠️ **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.

Device Selection API

The `getAgentConfig()` on page 131 and `setAgentConfig()` on page 136 methods should be implemented in the vendor connector. The `getAgentConfig()` method is invoked when the agent clicks the Agent Settings icon. It returns a `AgentConfigResult` on page 141 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 `GenericResult` object indicating whether the operation is successful. For example:

```
new GenericResult({
  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 on page 126 implementation. However, when using a desk phone, events and errors must be published from the vendor's code using `publishEvent()` on page 124 and `publishError()` on page 123.

For example, when clicking the mute button from a softphone, the `mute()` on page 134 method is called on the vendor's Connector API Interface on page 126 implementation and a promise is sent back to Salesforce.
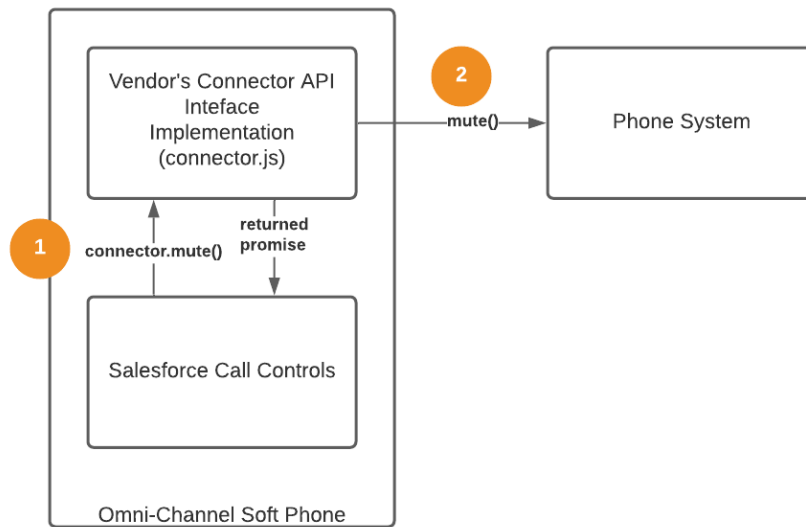
When using a hard phone, you must explicitly call `publishEvent()` on page 124 when the mute occurs.
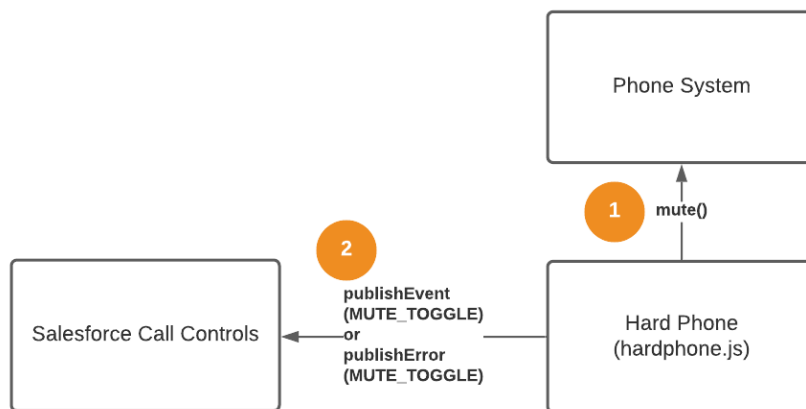
The `publishEvent()` on page 124 method can be called from the Connector API Interface implementation with many potential event types. The `publishError()` on page 123 method can be called for desk phone events for the same events. For example:

- 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 on page 124 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()` on page 124 method.

```
publishEvent({
  eventType: Constants.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()` on page 124, the connector can disable call controls if you prefer that agents 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` on page 162 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, agents 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 Connector API on page 120. Specifically, use the `publishEvent()` on page 124 method with the `CALL_STARTED` event type and a `CallResult` on page 148 payload.

This code sample is adapted from the Demo Connector on page 30 in GitHub (scv-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.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:

Get Started with the Service Cloud Voice Connector API

publishEvent

GenericResult

getAgentConfig

setAgentConfig

CallResult

PhoneCall

CHAPTER 15 Additional Info

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.

Einstein Conversation Insights (Call Coaching)

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

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 on page 176 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.
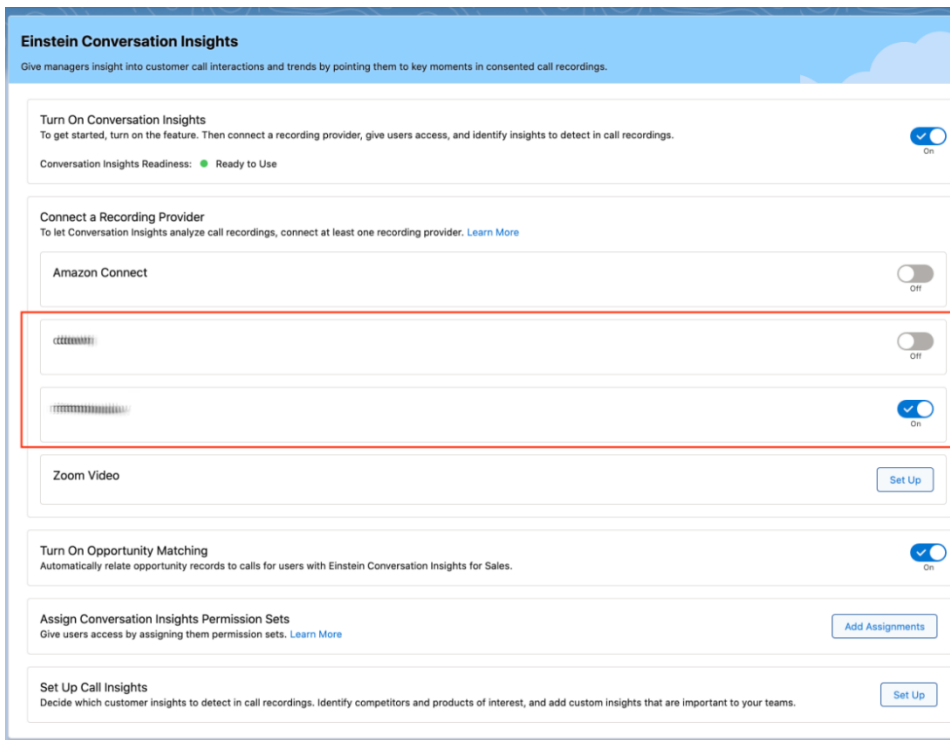
2. Implement `service_cloud_voice.RecordingMediaProvider` on page 185 in the Apex class. The method `getSignedUrls` is called by Salesforce with named credentials (if set up for the org) and a list of vendor call keys. The named credentials can be used to make a call out to the partner system for authentication. 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 an agent 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()` on page 131 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()` on page 131 method is implemented in the demo-scv-connector on page 30 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 agent 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:

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 an agent receiving or declining an inbound call.

Agent Accepts Inbound Call
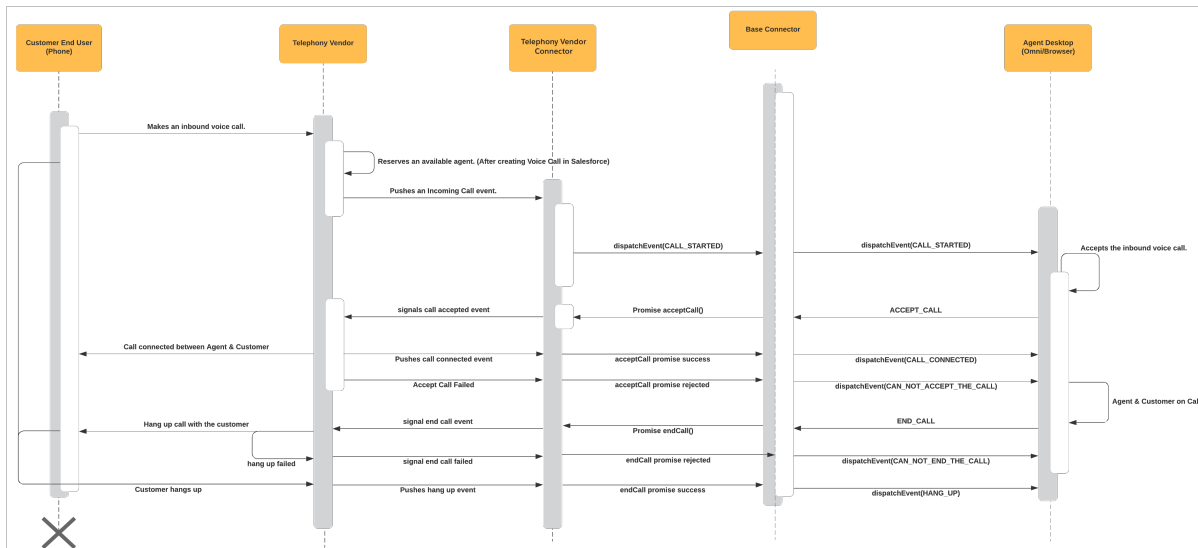
Agent Declines Inbound Call
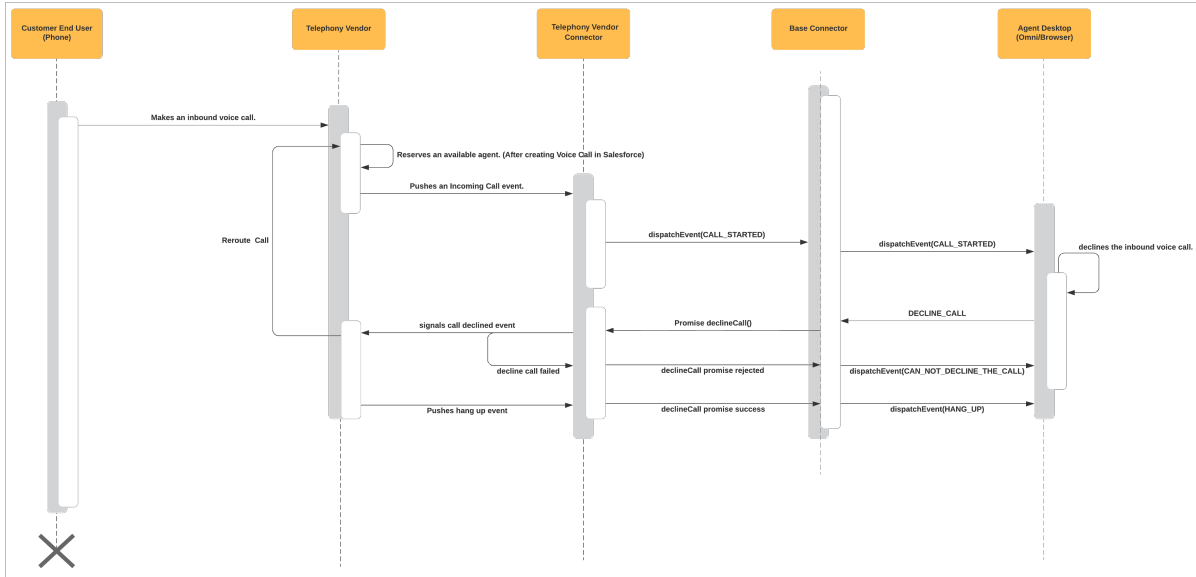
Agent Makes Outbound Call
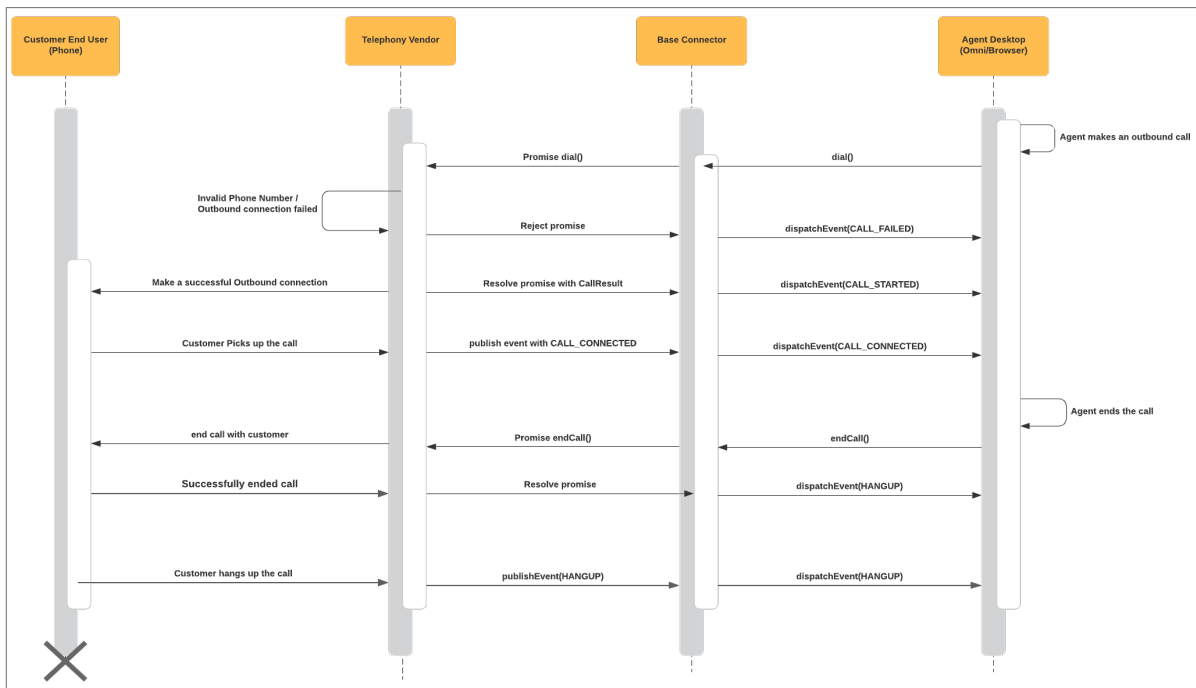
Agent Transfers a Call
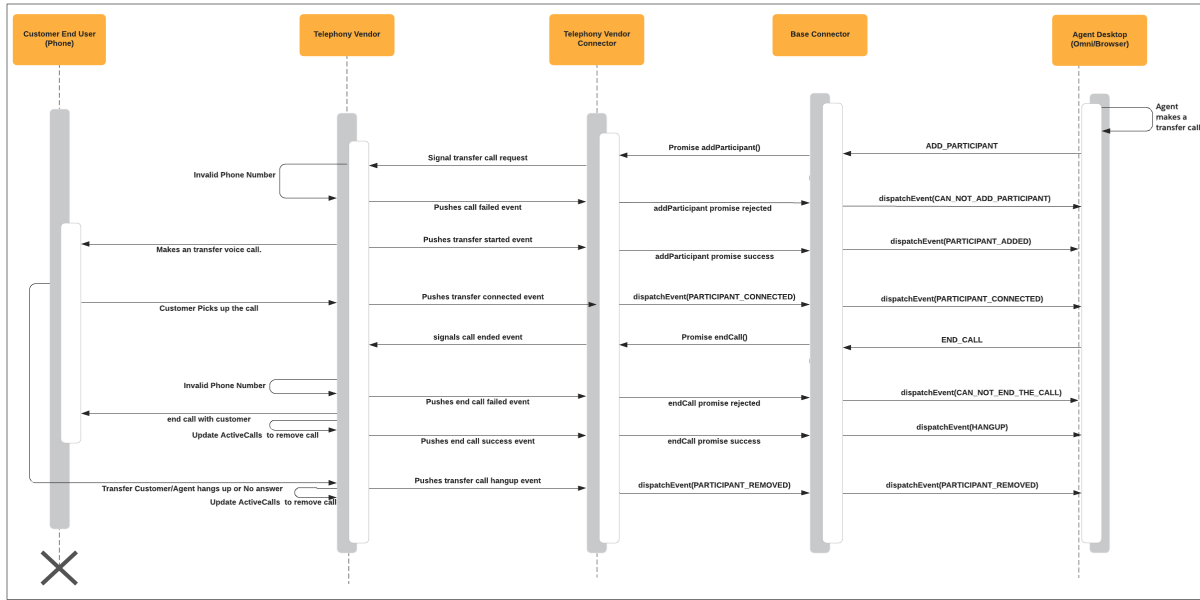
Line-Specific Controls with Service Cloud Voice for Partner Telephony

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 `CallInfo` on page 144 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.

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, Agent 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);
```

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

CHAPTER 16 Service Cloud Voice for Partner Telephony Connector API Reference

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

The Connector API contains two parts:

1. Connector API Base on page 122: The Connector API Base allows you to initialize the API, and to send information to Salesforce when events occur with your telephony system.

2. Connector API Interface on page 126: The Connector API Interface is what you implement in order to be notified when events occur within the Salesforce org.

When you initialize the Connector API using the Connector API Base on page 122, you pass it your implementation of the Connector API Interface on page 126.

SEE ALSO:

Get Started with the Service Cloud Voice Connector API

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

The Basics

To use the Connector API, you must call `initializeConnector()` (imported from `scv-connector-base`) as the first step during the window load process. This function is part of the Connector API Base on page 122 and ensures that Salesforce can reliably establish two-way communication with the connector without any timing issues.

The `initializeConnector()` function takes your Connector API Interface on page 126 implementation as a parameter. Salesforce holds onto the connector implementation object during the entire session and this object must implement the required Connector API Interface on page 126 functions. If the connector requires loading any resources, we recommend that you do this outside of the connector object, so that they can be loaded asynchronously without delaying the call to `initializeConnector()`. After you call `initializeConnector()`, Salesforce calls the `init()` on page 133 method on the Connector API Interface on page 126 with call center configuration information. This method returns a promise and can asynchronously load any resources before resolving the promise.

```html
<!-- Sample connector HTML -->
<html>
    <body>
        <div id="iframe" style="display: none;"></div>
    </body>
    <script type="module" src="connectordemo.js"></script>
</html>
```

```javascript
import { initializeConnector } from 'scv-connector-base';
import { Sdk } from './vendor-sdk';

const connector = new PartnerConnectorInterfaceImplementation();

// Calling the Connector API Base
window.addEventListener('load', () => {
    /**
    * initializeConnector should be called as early as possible
    * to allow Salesforce to reliably establish communication
    * with the connector
    */
    initializeConnector(connector);
});

// Connector API Interface Implementation
export class PartnerConnectorInterfaceImplementation {
    constructor(state) {
        this.sdk = new Sdk(state);
    }

    /**
     * Called by Salesforce to initialize the connector
```

```
     * @param {object} callCenterConfig - SFDC Contact Center Settings
     */
    init(callCenterConfig) {
        return new Promise(( resolve, reject ) => {
            this.sdk.performSSO(callCenterConfig).then( (success) => {
                if (success) resolve(new InitResult({}));
                else reject("Failed to perform SSO");
            });
        });
    }
}
```

Best Practices

Returning JavaScript promises in the connector greatly reduces the need to produce events or errors. Instead, the connector simply returns a resolved or rejected promise. A Connector API Interface implementation must return a promise that resolves to a value of a specific type.

Here are some sample promises along with type checking.

```
// Base connector
try {
    const result = await vendorConnector.setAgentStatus(agentStatus);

    // Validate result is of type GenericResult else throw error
    dispatchEvent(constants.EVENT_TYPE.SET_AGENT_STATUS_RESULT, { success });
} catch (e) {
    dispatchError(constants.ERROR_TYPE.CAN_NOT_SET_AGENT_STATUS, e);
}

// Connector
setAgentStatus(agentStatus) {
   return this.sdk.setAgentStatus(agentStatus).then((success) => {
        return new GenericResult({success});
   });
}

// Vendor
setAgentStaus(agentStatus) {
    return new Promise((resolve, reject) => {
        // Perform backend operations
        if (success) {
            resolve();
        } else {
            reject();
        }
    });
}
```

If you're using callbacks, you can convert them to promises as shown here:

```
// Callback
function setAgentStatus(agentStatus) {
    getAgent().setAgentStatus(agentStatus, {
```

121

```
        success: (success) => {
            getTelephonyEventEmitter().emit(Constants.EVENT_TYPE.SET_AGENT_STATUS_RESULT,
 { success });
        },
        failure: (response) => {
            getTelephonyEventEmitter().emit(Constants.EVENT_TYPE.SET_AGENT_STATUS_RESULT,
 { success: false });
        }
    });
}

// Promise
function setAgentStatus(agentStatus) {
    return new Promise((resolve, reject) => {
        getAgent().setAgentStatus(agentStatus, {
            success: (success) => {
                resolve({ success });
            },
            failure: (response) => {
                reject();
            }
        });
    });
}
```

SEE ALSO:

Service Cloud Voice for Partner Telephony Connector API Base

Service Cloud Voice for Partner Telephony Connector API Interface

Service Cloud Voice for Partner Telephony Connector API Base

The Connector API Base allows you to initialize the API, and to send information to Salesforce when events occur with your telephony system.

In addition to initializing the connector, the Connector API Base also provides two functions to publish events and errors to 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.

initializeConnector

Initializes the Salesforce Connector API Base.

publishError

Publishes a telephony error to Salesforce.

[publishEvent](#)

Publishes an event to Salesforce. The event payload is verified to ensure that it's the correct type before it's published. Use this method to publish an event from a desk phone.

SEE ALSO:

[Get Started with the Service Cloud Voice Connector API](#)

initializeConnector

Initializes the Salesforce Connector API Base.

Parameters

Field	Details
connector	**Type** object **Description** Your Connector API Interface on page 126 implementation.

SEE ALSO:

[Get Started with the Service Cloud Voice Connector API](#)

publishError

Publishes a telephony error to Salesforce.

Parameters

Field	Details
param	**Type** object **Description** Contains an event type and an error object.
param.eventType	**Type** string **Description** Event type for the error. Defaults to GENERIC_ERROR if none of the other event types apply. **Possible Values** • LOGIN_RESULT

Field	Details
	• LOGOUT_RESULT
	• CALL_STARTED
	• QUEUED_CALL_STARTED
	• CALL_CONNECTED
	• HANGUP
	• PARTICIPANT_CONNECTED
	• PARTICIPANT_ADDED
	• PARTICIPANTS_SWAPPED
	• PARTICIPANTS_CONFERENCED
	• MESSAGE
	• MUTE_TOGGLE
	• HOLD_TOGGLE
	• RECORDING_TOGGLE
	• UPDATE_AUDIO_STATS
	• SET_AGENT_STATUS
	• SUPERVISOR_CALL_STARTED on page 173
	• SUPERVISOR_CALL_CONNECTED on page 173
	• SUPERVISOR_HANGUP on page 173
	• GENERIC_ERROR
param.error	**Type** object **Description** Error object representing the error.

publishEvent

Publishes an event to Salesforce. The event payload is verified to ensure that it's the correct type before it's published. Use this method to publish an event from a desk phone.

Parameters

Field	Details
param	**Type** object **Description** Contains an event type and a payload.

Field	Details
`param.eventType`	**Type** string **Possible Values** - LOGIN_RESULT - LOGOUT_RESULT - CALL_STARTED - QUEUED_CALL_STARTED - CALL_CONNECTED - HANGUP - PARTICIPANT_CONNECTED - PARTICIPANT_ADDED - PARTICIPANTS_SWAPPED - PARTICIPANTS_CONFERENCED - MESSAGE - MUTE_TOGGLE - HOLD_TOGGLE - RECORDING_TOGGLE - UPDATE_AUDIO_STATS - SET_AGENT_STATUS - SUPERVISOR_CALL_STARTED on page 173 - SUPERVISOR_CALL_CONNECTED on page 173 - SUPERVISOR_HANGUP on page 173 - SUPERVISOR_BARGED_IN on page 173 - CALL_BARGED_IN on page 173 **Description** Event type to publish. If the agent status is offline, you shouldn't publish the following events: CALL_STARTED, QUEUED_CALL_STARTED, CALL_CONNECTED, HANGUP, PARTICIPANT_CONNECTED, PARTICIPANT_ADDED, PARTICIPANTS_SWAPPED, PARTICIPANTS_CONFERENCED, MESSAGE, MUTE_TOGGLE, HOLD_TOGGLE, RECORDING_TOGGLE, UPDATE_AUDIO_STATS. If you publish one of these events when an agent is offline, they're notified regardless of their status.
`param.payload`	**Type** object **Description** Payload for the event. The payload must be an object with the payload class associated with the EVENT_TYPE. Otherwise, the event isn't dispatched. - LOGIN_RESULT—GenericResult on page 156 - LOGOUT_RESULT—LogoutResult on page 159

Field	Details
	• CALL_STARTED—CallResult on page 148
	• QUEUED_CALL_STARTED—CallResult on page 148
	• CALL_CONNECTED—CallResult on page 148
	• HANGUP—HangupResult on page 157
	• PARTICIPANT_CONNECTED—ParticipantResult on page 161
	• PARTICIPANT_ADDED—ParticipantResult on page 161
	• PARTICIPANTS_SWAPPED—HoldToggleResult on page 157
	• PARTICIPANTS_CONFERENCED—HoldToggleResult on page 157
	• MESSAGE—object
	• MUTE_TOGGLE—MuteToggleResult on page 160
	• HOLD_TOGGLE—HoldToggleResult on page 157
	• RECORDING_TOGGLE—RecordingToggleResult on page 165
	• UPDATE_AUDIO_STATS—AudioStats on page 142
	• SET_AGENT_STATUS—AgentStatusInfo on page 143
	• SUPERVISOR_CALL_STARTED on page 173—SuperviseCallResult on page 170
	• SUPERVISOR_CALL_CONNECTED on page 173—SuperviseCallResult on page 170
	• SUPERVISOR_HANGUP on page 173—SuperviseCallResult on page 170
	• CALL_BARGED_IN on page 173—SuperviseCallResult on page 170

Example

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

Service Cloud Voice for Partner Telephony Connector API Interface

The Connector API Interface is what you implement in order to be notified when events occur within the Salesforce 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.

acceptCall
Accept a call.

addParticipant
Add a participant to a call.

conference
Create a conference call from multiple calls.

declineCall

Decline a call.

dial

Dial a contact.

endCall

End a call.

getActiveCalls

Get a list of active call.

getAgentConfig

Get configuration information for the agent.

getCapabilities

Get the partner telephony capabilities.

getPhoneContacts

Get a list of phone contacts and contact types. This function is called when speed dial is clicked in order to get the agent's phone contacts.

getSignedRecordingUrl

Get the signed recording URL.

handleMessage

Handle a message from a Lightning Component.

hold

Put a call on hold.

init

Initialize the Connector API Interface.

logout

Log out from the telephony system. You must implement this method, which is called when an agent logs out of the phone system from Omni-Channel.

mute

Mute a call.

pauseRecording

Pause the recording of a call.

resume

Resume a call that is on hold.

resumeRecording

Resume recording a call.

sendDigits

Send digits on an existing call.

setAgentConfig

Set configuration information for the agent.

setAgentStatus

Set status information for the agent.

Swap one call with another. The participant on hold is resumed and the resumed participant is placed on hold.

Unmute a call.

Wrap up a call. The implementation of this method should remove the agent from wrap-up and return them to the queue for receiving calls.

SEE ALSO:

acceptCall

Accept a call.

Signature

```
acceptCall(call) → Promise.<CallResult on page 148>
```

Parameters

Field	Details
call	**Type** PhoneCall on page 162 **Description** Reserved for future use. Currently, this value isn't used and is always null.

addParticipant

Add a participant to a call.

📝 Note: As with outbound calls, you must create a voice call when adding a new participant. To learn more, see Create Voice Call API.

Signature

```
addParticipant(contact, call) → Promise.<ParticipantResult on page 161>
```

Parameters

Field	Details
contact	**Type** Contact on page 154 **Description** The contact to add.
call	**Type** PhoneCall on page 162 **Description** The call to be accepted.
isBlindTransfer	**Type** boolean **Description** Indicates whether this participant is a blind transfer.

conference

Create a conference call from multiple calls.

Signature

```
conference(calls) → Promise.<ConferenceResult on page 153>
```

Parameters

Field	Details
calls	**Type** Array.<PhoneCall on page 162> **Description** The calls to conference together. Currently, this function only supports conferencing two calls together.

declineCall

Decline a call.

Signature

```
declineCall(call) → Promise.<CallResult on page 148>
```

Parameters

Field	Details
call	**Type** PhoneCall on page 162 **Description** Reserved for future use. Currently, this value isn't used and is always null.

dial

Dial a contact.

Signature

```
dial(contact) → Promise.<GenericResult on page 156>
```

Parameters

Field	Details
contact	**Type** Contact on page 154 **Description** The contact to dial.

endCall

End a call.

Signature

```
endcall(call) → Promise.<HangupResult on page 157>
```

Parameters

Field	Details
call	**Type** PhoneCall on page 162

Field	Details
	Description
	Reserved for future use. Currently, this value isn't used and is always null.

getActiveCalls

Get a list of active call.

Signature

```
getActiveCalls() → Promise.<ActiveCallsResult on page 140>
```

getAgentConfig

Get configuration information for the agent.

Signature

```
getAgentConfig() → Promise.<AgentConfigResult on page 141>
```

getCapabilities

Get the partner telephony capabilities.

Signature

```
getCapabilities() → Promise.<CapabilitiesResult on page 148>
```

getPhoneContacts

Get a list of phone contacts and contact types. This function is called when speed dial is clicked in order to get the agent's phone contacts.

Signature

```
getPhoneContacts() → Promise.<PhoneContactsResult on page 164>
```

getSignedRecordingUrl

Get the signed recording URL.

Note: You need to enable the signedUrl capability in the agent config to use this feature.

Signature

```
getSignedRecordingUrl(recordingUrl, vendorCallKey, callId) →
Promise.<SignedRecordingUrlResult on page 166>
```

Parameters

Field	Details
recordingUrl	**Type** string **Description** The URL for the recording.
vendorCallKey	**Type** string **Description** Vendor call key.
callId	**Type** string **Description** ID of the call.

handleMessage

Handle a message from a Lightning Component.

Signature

```
handleMessage(message)
```

Parameters

Field	Details
message	**Type** object **Description** The message to be handled.

hold

Put a call on hold.

Signature

`hold(call) → Promise.<`HoldToggleResult` on page 157>`

Parameters

Field	Details
`call`	**Type** PhoneCall on page 162 **Description** Reserved for future use. Currently, this value isn't used and is always null.

init

Initialize the Connector API Interface.

Signature

`init(connectorConfig) → Promise.<`InitResult` on page 158>`

Parameters

Field	Details
`connectorConfig`	**Type** object **Description** This object includes the entire call center configuration along with the following values: • scvVendorLoginUrl • isACWAllowed • callCenterId • organizationId • scrtUrl • phoneServiceChannelName • callCenterType • phoneServiceChannelId • orgDomainName • userName

Field	Details
	• userId

SEE ALSO:

[Get Started with the Service Cloud Voice Connector API](#)

logout

Log out from the telephony system. You must implement this method, which is called when an agent logs out of the phone system from Omni-Channel.

Signature

```
logout() → Promise.<GenericResult on page 156>
```

SEE ALSO:

[Develop a Telephony System Login Page](#)

mute

Mute a call.

Signature

```
mute() → Promise.<MuteToggleResult on page 160>
```

pauseRecording

Pause the recording of a call.

Signature

```
pauseRecording(call) → Promise.<RecordingToggleResult on page 165>
```

Parameters

Field	Details
call	**Type** [PhoneCall](#) on page 162 **Description** Reserved for future use. Currently, this value isn't used and is always null.

resume

Resume a call that is on hold.

Signature

`resume(call)` → `Promise.<`HoldToggleResult` on page 157>`

Parameters

Field	Details
`call`	**Type** PhoneCall on page 162 **Description** Reserved for future use. Currently, this value isn't used and is always null.

resumeRecording

Resume recording a call.

Signature

`resumeRecording(call)` → `Promise.<`RecordingToggleResult` on page 165>`

Parameters

Field	Details
`call`	**Type** PhoneCall on page 162 **Description** Reserved for future use. Currently, this value isn't used and is always null.

sendDigits

Send digits on an existing call.

Signature

`sendDigits(digits)` → `Promise`

Parameters

Field	Details
digits	**Type** string **Description** The digits to send.

setAgentConfig

Set configuration information for the agent.

Signature

setAgentConfig(config) → Promise.<GenericResult on page 156>

Parameters

Field	Details
config	**Type** object **Description** Agent configuration info.
config.hasMute	**Type** boolean **Description** Indicates whether mute is supported.
config.hasRecord	**Type** boolean **Description** Indicates whether recording is supported.
config.hasMerge	**Type** boolean **Description** Indicates whether merge is supported.
config.hasSwap	**Type** boolean

Field	Details
	Description Indicates whether swap is supported.
config.phones	**Type** Array<string> **Description** Optional. Supported phone types. Phone type value can be "SOFT_PHONE" or "DESK_PHONE".
config.selectedPhone	**Type** string **Description** Selected phone type. Phone type value can be "SOFT_PHONE" or "DESK_PHONE".
config.supportsMos	**Type** boolean **Description** Indicates whether Mean Opinion Score (MOS) is supported.

setAgentStatus

Set status information for the agent.

Signature

setAgentStatus(agentStatus, agentStatusInfo) → Promise.<GenericResult on page 156>

Parameters

Field	Details
agentStatus	**Type** string **Description** The agent status, as configured in Salesforce.
agentStatusInfo	**Type** object **Description** Additional information about the agent status.
enqueueNextState	**Type** boolean

137

Field	Details
	Description Indicates whether the state should be enqueued, which updates the agent's status after a call ends. Defaults to `false`.

swap

Swap one call with another. The participant on hold is resumed and the resumed participant is placed on hold.

Signature

```
swap(call1, call2) → Promise.<HoldToggleResult on page 157>
```

Parameters

Field	Details
swap1	**Type** PhoneCall on page 162 **Description** The first call to swap.
swap2	**Type** PhoneCall on page 162 **Description** The second call to swap.

unmute

Unmute a call.

Signature

```
unmute() → Promise.<MuteToggleResult on page 160>
```

wrapUpCall

Wrap up a call. The implementation of this method should remove the agent from wrap-up and return them to the queue for receiving calls.

Signature

```
wrapUpCall()
```

Service Cloud Voice for Partner Telephony Connector API Objects

These objects are used by the Connector API Interface and the Connector API Base.

🛑 **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.

ActiveCallsResult

Represents the result when getting a list of active calls.

AgentConfigResult

Represents the result when getting agent phone configuration information.

AudioStatsElement

Represents audio stats element, which can include audio input channel stats and audio output channel stats.

AudioStats

Represents audio stats, which contains an array of `AudioStatsElement` objects.

AgentStatusInfo

Represents the agent status.

CallInfo

Represents the call info for the transfer call using key value pairs in a JSON object.

CallResult

Represents the result type for call-related functions.

CapabilitiesResult

Represents the result type for the agent capabilities.

customError

Represents a custom error.

ConferenceResult

Represents the result for the `conference()` function.

Contact

Represents information about a phone call.

GenericResult

Represents a generic result type for a Connector API function.

HangupResult

Represents a result type for an ended call.

HoldToggleResult

Represents the result when toggling the hold state for a call.

InitResult

Represents the result type for the init call.

LogoutResult

Represents a logout result type.

MuteToggleResult

Represents the result when toggling the mute state for a call.

ParticipantResult

Represents the result for the `addParticipant()` function.

Phone

Represents information about a phone.

PhoneCall

Represents information about a phone call.

PhoneContactsResult

Represents the result when getting a list of phone contacts.

RecordingToggleResult

Represents the result when toggling the recording for a call.

SignedRecordingUrlResult

Represents the result when getting the signed recording URL.

StatsInfo

Representing stats for either an audio input channel or an audio output channel. This object is used to calculate the MOS (Mean Opinion Score).

SEE ALSO:

Get Started with the Service Cloud Voice Connector API

Service Cloud Voice for Partner Telephony Connector API Base

Service Cloud Voice for Partner Telephony Connector API Interface

ActiveCallsResult

Represents the result when getting a list of active calls.

Constructor

```
new ActiveCallsResult(param)
```

Functions Where Used

Parameters

Field	Details
param	**Type** object **Description** The hold toggle result.

Field	Details
param.activeCalls	**Type** Array<PhoneCall on page 162> **Description** A list of phone calls that are active.

AgentConfigResult

Represents the result when getting agent phone configuration information.

Constructor

```
new AgentConfigResult(param)
```

Functions Where Used

- getAgentConfig() on page 131

Objects Referenced

- Phone on page 161

Parameters

Field	Details
param	**Type** object **Description** The agent config result.
param.phones	**Type** Array<string> **Description** Optional. Supported phone types. Phone type value can be "SOFT_PHONE" or "DESK_PHONE".
param.selectedPhone	**Type** Phone on page 161 **Description** Selected phone object, which includes the type and number.

Example

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

AudioStatsElement

Represents audio stats element, which can include audio input channel stats and audio output channel stats.

Constructor

```
new AudioStatsElement(param)
```

Objects Where Used

- AudioStats on page 142

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with this object.
param.inputChannelStats	**Type** StatsInfo on page 167 **Description** The audio input channel stats info.
param.outputChannelStats	**Type** StatsInfo on page 167 **Description** The audio output channel stats info.

AudioStats

Represents audio stats, which contains an array of `AudioStatsElement` objects.

Constructor

```
new AudioStats(param)
```

Functions Where Used

- publishEvent() on page 124

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with this object.
param.stats	**Type** Array<AudioStatsElement on page 142> **Description** An array of audio stats elements.
param.callId	**Type** string **Description** The unique call ID.
param.isAudioStatsCompleted	**Type** boolean **Description** This value is true when the audio stats are completed. When true, Salesforce calculates the Mean Opinion Score (MOS) and updates the VoiceCall record based on the callId. For more information, see Mean Opinion Score (MOS) on page 87.

AgentStatusInfo

Represents the agent status.

Constructor

new AgentStatusInfo(param)

Functions Where Used

- publishEvent() on page 124

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the agent status info.
param.statusId	**Type** string **Description** The Salesforce status ID.

CallInfo

Represents the call info for the transfer call using key value pairs in a JSON object.

Constructor

`new CallInfo(param)`

Objects Where Used

- **PhoneCall** on page 162
- **ParticipantResult** on page 161

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the call info.
param.acceptEnabled	**Type** boolean **Description** Indicates whether the accept call button is enabled.
param.addCallerEnabled	**Type** boolean

Field	Details
	Description Indicates whether the add caller button is enabled.
param.callStateTimestamp	**Type** date **Description** The call timestamp.
param.conferenceEnabled	**Type** boolean **Description** Indicates whether the merge button is enabled.
param.declineEnabled	**Type** boolean **Description** Indicates whether the decline call button is enabled.
param.extensionEnabled	**Type** boolean **Description** Indicates whether an extension is enabled.
param.holdEnabled	**Type** boolean **Description** Indicates whether the hold button is enabled.
param.initialCallId	**Type** string **Description** Initial voice call ID.
param.isBargeable	**Type** boolean **Description** Reserved for future use.
param.isMuted	**Type** boolean **Description** Indicates whether the call is on mute.

Field	Details
param.isOnHold	**Type** boolean **Description** Indicates whether the call is on hold.
param.isRecordingPaused	**Type** boolean **Description** Indicates whether the recording is paused.
param.isReplayable	**Type** boolean **Description** Indicates whether the call is replayable.
param.isSoftphoneCall	**Type** boolean **Description** Indicates whether this call is made with a softphone.
param.muteEnabled	**Type** boolean **Description** Indicates whether the mute button is enabled.
param.recordEnabled	**Type** boolean **Description** Indicates whether the toggle recording button is enabled.
param.removeParticipantVariant	**Type** string **Description** Describes the remove participant button functionality. 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.
param.showMuteButton	**Type** boolean

Field	Details
	Description Indicates whether to show the mute button. Default value is true.
param.showRecordButton	**Type** boolean
	Description Indicates whether to show the record button. Default value is true.
param.showAddCallerButton	**Type** boolean
	Description Indicates whether to show the add caller button. Default value is true.
param.showAddBlindTransferButton	**Type** boolean
	Description Indicates whether to show the blind transfer button. Default value is true.
param.showMergeButton	**Type** boolean
	Description Indicates whether to show the merge button. Default value is true.
param.showSwapButton	**Type** boolean
	Description Indicates whether to show the swap button. Default value is true.
param.swapEnabled	**Type** boolean
	Description Indicates whether the swap button is enabled.
param.queueId	**Type** string
	Description The unique ID of the queue taken from the telephony provider's side.
param.queueName	**Type** string
	Description The name of the queue that appears in the call record.

Field	Details
param.queueTimestamp	**Type** date **Description** The date and time (in UTC) when the call entered the queue. Measured in milliseconds since the Unix epoch.

CallResult

Represents the result type for call-related functions.

Constructor

```
new CallResult(param)
```

Functions Where Used

- acceptCall() on page 128
- addParticipant() on page 128
- declineCall() on page 129
- publishEvent() on page 124

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the call result.
param.call	**Type** PhoneCall on page 162 **Description** The call associated with this result.

CapabilitiesResult

Represents the result type for the agent capabilities.

Constructor

```
new CapabilitiesResult(param)
```

Functions Where Used

- getCapabilities()

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the capabilities result.
param.debugEnabled	**Type** boolean **Description** Indicates whether the agent can download logs.
param.hasAgentAvailability	**Type** boolean **Description** Indicates whether the connector provides agent availability for phone book contacts. If this parameter is `false`, Salesforce tries to calculate availability for phone book contacts of type Agent and the value specified in `Contact.availability` is ignored.
param.hasBlindTransfer	**Type** boolean **Description** Indicates whether the vendor supports blind transfer. The default value is `false`.
param.hasContactSearch	**Type** boolean **Description** Indicates whether the vendor supports contact search.
param.hasMerge	**Type** boolean **Description** Indicates whether merge is supported.

Field	Details
`param.hasMute`	**Type** boolean **Description** Indicates whether mute is supported.
`param.hasPendingStatusChange`	**Type** boolean **Description** Indicates whether the pending status change behavior is enabled.
`param.hasPhoneBook`	**Type** boolean **Description** Indicates whether agent-to-agent and agent-to-queue calls are enabled in the Omni-Channel widget. The default value is `false`.
`param.hasQueueWaitTime`	**Type** boolean **Description** Indicates whether Salesforce or the partner telephony provider calculates estimated wait time values. If this parameter is `false`, Salesforce calculates the estimated wait times. If this parameter is `true`, the telephony provider calculates the estimated wait times. The default value is `false`.
`param.hasRecord`	**Type** boolean **Description** Indicates whether recording is supported.
`param.hasSupervisorBargeIn`	**Type** boolean **Description** Indicates whether the supervisor can barge in. The default value is `false`.
`param.hasSupervisorListenIn`	**Type** boolean **Description** Indicates whether the supervisor can listen in.
`param.hasSwap`	**Type** boolean

Field	Details
	Description
	Indicates whether swap is supported.
`param.hasTransferToOmniFlow`	**Type**
	boolean
	Description
	Indicates whether the call can be transferred to an Omni-Channel flow. The default value is `false`.
`param.signedRecordingDuration`	**Type**
	string
	Description
	The signed recording duration.
`param.signedRecordingUrl`	**Type**
	string
	Description
	The signed recording URL.
`param.supportsMos`	**Type**
	boolean
	Description
	Indicates whether the vendor supports MOS scores.

Example

```
new CapabilitiesResult({
    hasMute: true,
    hasMerge: true,
    hasRecord: true,
    hasSwap:  true,
    hasSignedRecordingUrl: false,
    hasContactSearch: true,
    supportsMos: true,
    hasAgentAvailability: true,
    hasQueueWaitTime: true,
    hasSupervisorListenIn: true,
    hasSupervisorBargeIn: false,
    debugEnabled: true,
    hasBlindTransfer: true,
    hasTransferToOmniFlow: true,
    signedRecordingUrl: null,
    signedRecordingDuration: null
});
```

customError

Represents a custom error.

Constructor

```
new CustomError(param)
```

Functions Where Used

Parameters

Field	Details
param	**Type** object **Description** Contains the parameters associated with the custom error message.
param.labelName	**Type** string

Field	Details
	Description The name of the custom label in your org. This name is defined when you [create the custom label](#) for the error message.
`param.namespace`	**Type** CallAttributes **Description** The alphanumeric identifier of your Salesforce org's namespace prefix. The namespace is a part of the label that makes up the required full name used for managed packages. The full name is `<namespace.labelName>`. If you don't have a namespace prefix, set this value to `C` .
`param.message`	**Type** [CallInfo](#) on page 144 **Description** Optional. Additional details about the message for logging purposes. This field is optional and is not exposed.

ConferenceResult

Represents the result for the `conference()` function.

Constructor

`new ConferenceResult(isThirdPartyOnHold, isCustomerOnHold)`

Functions Where Used

- `conference()` on page 129

Parameters

Field	Details
`isThirdPartyOnHold`	**Type** boolean **Description** Indicates whether the third-party participant of the call is on hold.
`isCustomerOnHold`	**Type** boolean

Field	Details
	Description Indicates whether the customer is on hold.

Contact

Represents information about a phone call.

Constructor

```
new Contact(param)
```

Functions Where Used

Objects Where Used

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with a contact.
param.id	**Type** string **Description** Optional. The unique contact ID.
param.type	**Type** string **Description** Optional. The type of the contact. One of these values: • PhoneBook • Queue

Field	Details
	• PhoneNumber
	• Agent
`param.name`	**Type**
	string
	Description
	Optional. The label for this contact to be displayed in the UI.
`param.phoneNumber`	**Type**
	string
	Description
	Optional. The phone number associated with this contact.
`param.prefix`	**Type**
	string
	Description
	Optional. Any prefix to be dialed before dialing the number (for example, "+1").
`param.extension`	**Type**
	string
	Description
	Optional. Any extension to be dialed after dialing the number.
`param.endpointARN`	**Type**
	string
	Description
	Optional.
`param.queue`	**Type**
	string
	Description
	Optional. The unique ID of the contact center queue.
`param.availability`	**Type**
	string (Constants.AGENT_AVAILABILITY)
	Description
	Optional. Value for agent availability. This is only used when `AgentConfigResult.hasAgentAvailability` on page 141 is set to `true`. Allowed values are:
	• AVAILABLE
	• BUSY
	• OFFLINE

Field	Details
param.queueWaitTime	**Type** string **Description** Optional. The estimated queue wait time (measured in seconds) calculated by the telephony provider.

GenericResult

Represents a generic result type for a Connector API function.

Constructor

```
new GenericResult(param)
```

Functions Where Used

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the logout result.
param.success	**Type** boolean **Description** Indicates whether the function succeeded.
param.loginFrameHeight	**Type** number **Description** Optional. Custom login page iframe height.

Example

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

HangupResult

Represents a result type for an ended call.

Constructor

```
new HangupResult(param)
```

Functions Where Used

- `endcall()` on page 130

Parameters

Field	Details
`param`	**Type** object **Description** Parameters associated with the result.
`param.calls`	**Type** Array<PhoneCall on page 162> **Description** One or more phone calls that ended. This value can be more than one when an agent ends the call or hangs up.

HoldToggleResult

Represents the result when toggling the hold state for a call.

Constructor

```
new HoldToggleResult(param)
```

Functions Where Used

- `hold()` on page 133
- `resume()` on page 135

Parameters

Field	Details
param	**Type** object **Description** The hold toggle result.
param.isThirdPartyOnHold	**Type** boolean **Description** Indicates whether a transferred call is on hold.
param.isCustomerOnHold	**Type** boolean **Description** Indicates whether the customer call is on hold.
param.calls	**Type** Array<PhoneCall on page 162> **Description** Optional. A list of phone calls associated with this result.

InitResult

Represents the result type for the init call.

Constructor

```
new InitResult(param)
```

Functions Where Used

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the init result.
param.showLogin	**Type** boolean **Description** Optional. Indicates whether the custom login page is required.
param.loginFrameHeight	**Type** number **Description** Optional. Custom login page iframe height.

SEE ALSO:

[Get Started with the Service Cloud Voice Connector API](#)

LogoutResult

Represents a logout result type.

Constructor

```
new LogoutResult(param)
```

Functions Where Used

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the result.

Field	Details
`param.success`	**Type** boolean **Description** Indicates whether the function succeeded.
`param.loginFrameHeight`	**Type** number **Description** Optional. The height of the login/logout frame. If no value is provided, a default value of 350 is used.

MuteToggleResult

Represents the result when toggling the mute state for a call.

Constructor

`new MuteToggleResult(param)`

Functions Where Used

- `mute()` on page 134
- `unmute()` on page 138
- `publishEvent()` on page 124

Parameters

Field	Details
`param`	**Type** object **Description** The mute toggle result.
`param.isMuted`	**Type** boolean **Description** Indicates whether the call is muted.

ParticipantResult

Represents the result for the `addParticipant()` function.

Constructor

```
new ParticipantResult(initialCallHasEnded, callInfo, phoneNumber, callId)
```

Functions Where Used

- `addParticipant()` on page 128
- `publishEvent()` on page 124

Parameters

Field	Details
initialCallHasEnded	**Type** boolean **Description** The parent call value for `callAttributes.initialCallHasEnded`.
callInfo	**Type** CallInfo on page 144 **Description** Call info for the transfer call using key value pairs in a JSON object.
phoneNumber	**Type** string **Description** Phone number of the new participant.
callId	**Type** string **Description** Optional. New transfer call ID.

Phone

Represents information about a phone.

Constructor

```
new Phone(param)
```

Objects Where Used

-

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with a phone.
param.type	**Type** string **Description** The type of phone. Phone type value can be "SOFT_PHONE" or "DESK_PHONE".
param.number	**Type** string **Description** The phone number associated with this phone. This value is optional when the type is SOFT_PHONE and mandatory when the type is DESK_PHONE.

PhoneCall

Represents information about a phone call.

Constructor

```
new PhoneCall(param)
```

Objects Where Used

-
-

Parameters

Field	Details
param	**Type** object

Field	Details
	Description Parameters associated with a phone call.
`param.agentStatus`	**Type** string **Description** Optional. Status of the agent on the call.

`param.callAttributes`	**Type**
	CallAttributes
	
	Description
	Attributes associated with the call. The values in this object are immutable and can't be updated by the partner.
	
	<pre>callAttributes: {
	  voiceCallId: string,
	  participantType: string,
	  dialerType: string,
	  parentId: string
	}</pre>

`param.callInfo`	**Type**
	CallInfo on page 144
	
	Description
	Optional. Call info using key value pairs in a JSON object.
	
	<pre>callInfo: {
	  acceptEnabled: boolean,
	  declineEnabled: boolean,
	  holdEnabled: boolean,
	  muteEnabled: boolean,
	  extensionEnabled: boolean,
	  swapEnabled: boolean,
	  conferenceEnabled: boolean,
	  extensionEnabled: boolean,
	  recordEnabled: boolean,
	  addCallerEnabled: boolean
	}</pre>

`param.callType`	**Type** string **Description** Optional. The type of call. One of these values: • Inbound • Outbound • Callback

Field	Details
	• AddParticipant
param.closeCallOnError	**Type** boolean **Description** Optional. Whether the call was closed for an error.
param.contact	**Type** Contact on page 154 **Description** Optional. The call target or contact.
param.phoneNumber	**Type** string **Description** Optional. The phone number associated with this call; usually an external number.
param.reason	**Type** string **Description** Optional. Reason for phone call.
param.state	**Type** string **Description** Optional. The state of the call. For example: ringing, connected, declined, failed.

PhoneContactsResult

Represents the result when getting a list of phone contacts.

Constructor

```
new PhoneContactsResult(param)
```

Functions Where Used

• getAgentConfig() on page 131

Parameters

Field	Details
param	**Type** object **Description** The phone contacts result.
param.contacts	**Type** Array<Contact on page 154> **Description** A list of contacts.
param.contactTypes	**Type** Array of strings **Description** A list of contact types. Possible values: • PhoneBook • Queue • PhoneNumber • Agent • Flow

RecordingToggleResult

Represents the result when toggling the recording for a call.

Constructor

```
new RecordingToggleResult(param)
```

Functions Where Used

- pauseRecording() on page 134
- resumeRecording() on page 135
- publishEvent() on page 124

Parameters

Field	Details
`param`	**Type** object **Description** The recording toggle result.
`param.isRecordingPaused`	**Type** boolean **Description** Indicates whether the recording is paused.
`param.contactId`	**Type** string **Description** *Not needed for Service Cloud Voice with Partner Telephony.*
`param.initialContactId`	**Type** string **Description** *Not needed for Service Cloud Voice with Partner Telephony.*
`param.instanceId`	**Type** string **Description** *Not needed for Service Cloud Voice with Partner Telephony.*
`param.region`	**Type** string **Description** *Not needed for Service Cloud Voice with Partner Telephony.*

SignedRecordingUrlResult

Represents the result when getting the signed recording URL.

Constructor

`new SignedRecordingUrlResult(param)`

Functions Where Used

-

Parameters

Field	Details
param	**Type** object **Description** The signed recording result.
param.success	**Type** boolean **Description** Indicates whether the function succeeded.
param.url	**Type** string **Description** Optional. The URL for the signed recording.
param.duration	**Type** number **Description** Optional. Recording duration in seconds.

StatsInfo

Representing stats for either an audio input channel or an audio output channel. This object is used to calculate the MOS (Mean Opinion Score).

Constructor

```
new StatsInfo(param)
```

Objects Where Used

-

Parameters

Field	Details
`param`	**Type** object **Description** Parameters associated with this object.
`param.packetsCount`	**Type** Number **Description** The packet count
`param.packetsLost`	**Type** Number **Description** The number of packets lost.
`param.jitterBufferMillis`	**Type** Number **Description** The jitter buffer time in milliseconds.
`param.roundTripTimeMillis`	**Type** Number **Description** The network round-trip time in milliseconds.

Service Cloud Voice for Partner Telephony Supervisor API

This section describes the Supervisor API, which is a part of the Connector API that is intended for supervisor functionality.

🛑 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.

superviseCall
Supervise a call. The supervised call is connected and muted. No other call controls exist other than "Leave Call".

supervisorBargeIn
Barge in on a call as a supervisor. The supervisor becomes a new participant in the call. The supervisor is unmuted and has the usual call controls (mute, hold, and recording toggles).

SuperviseCallResult
Represents the result when supervising a call.

SupervisedCallInfo

Represents information about a supervised call.

supervisorDisconnect

Disconnect a supervised call.

SupervisorHangupResult

Represents a result type for an ended supervisor call.

Supervisor Events

These events can be used by a softphone (sent from the base connector automatically) or by using the publishEvent API (when on a desk phone). These events can only be sent if the agent has the capability `isSupervisor`. They're used by Salesforce to update the Supervisor Call Monitor UI.

SEE ALSO:

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

publishEvent

superviseCall

Supervise a call. The supervised call is connected and muted. No other call controls exist other than "Leave Call".

The vendor call ID is part of the SuperviseCallResult on page 170.

📝 **Note:** This method can only be invoked if the agent has the capability `hasSupervisorListenIn`.

Signature

```
superviseCall(supervisedCallInfo) → Promise.<SuperviseCallResult on page 170>
```

Parameters

Field	Details
supervisedCallInfo	**Type** SupervisedCallInfo **Description** Info about the call to be supervised.

supervisorBargeIn

Barge in on a call as a supervisor. The supervisor becomes a new participant in the call. The supervisor is unmuted and has the usual call controls (mute, hold, and recording toggles).

📝 **Note:** This method can only be invoked if the agent has the capability `hasSupervisorBargeIn`.

Signature

```
supervisorBargeIn(supervisedCallInfo) → Promise.<SupervisedCallResult on page 170>
```

Parameters

Field	Details
supervisedCallInfo	**Type** SupervisedCallInfo **Description** Info about the call to be supervised.

SuperviseCallResult

Represents the result when supervising a call.

Constructor

```
new SuperviseCallResult(param)
```

Functions Where Used

- superviseCall() on page 169
- publishEvent() on page 124

Parameters

Field	Details
param	**Type** object **Description** The supervise call result.
param.call	**Type** PhoneCall on page 162 **Description** The call to be supervised.

SupervisedCallInfo

Represents information about a supervised call.

Constructor

```
new SupervisedCallInfo(param)
```

Functions Where Used

- supervisorBargeIn() on page 169

Parameters

Field	Details
param	**Type** object **Description** Information about the supervised call.
param.callId	**Type** string **Description** Required. The unique vendor call ID.
param.voiceCallId	**Type** string **Description** The unique voice call ID (`voiceCallId`).
param.callType	**Type** string **Description** The type of call. Possible values include: - Inbound - Outbound - Callback - AddParticipant - Transfer
param.from	**Type** string **Description** The phone number in the "From" field of the call for inbound calls.
param.to	**Type** string

Field	Details
	Description
	The phone number in the "To" field of the call for outbound calls.
`param.supervisorName`	**Type**
	string
	Description
	The name of the supervisor who is barging in to the call. The name is shown to the supervised agent.
`param.isBargedIn`	**Type**
	boolean
	Description
	Indicates whether the supervisor has barged in (true) or is listening in (false) to the call.

supervisorDisconnect

Disconnect a supervised call.

Signature

```
supervisorDisconnect(supervisedCallInfo) → Promise.<SupervisorHangupResult on page 172>
```

Parameters

Field	Details
`supervisedCallInfo`	**Type**
	object
	Description
	Information about the call to be supervised. This value is a JavaScript object with the following properties:
	• voiceCallId — The unique voice call ID.
	• callId — The unique vendor call ID.
	• to — The phone number in the "To" field of the call for outbound calls.
	• from — The phone number in the "From" field of the call for inbound calls.
	• callType — The type of call. Possible values are `Inbound`, `Outbound`, and `Transfer`.

SupervisorHangupResult

Represents a result type for an ended supervisor call.

172

Constructor

```
new SupervisorHangupResult(param)
```

Functions Where Used

- `supervisorDisconnect()` on page 172

Parameters

Field	Details
param	**Type** object **Description** Parameters associated with the result.
param.calls	**Type** Array<PhoneCall on page 162> **Description** One or more phone calls that ended.

Supervisor Events

These events can be used by a softphone (sent from the base connector automatically) or by using the publishEvent API (when on a desk phone). These events can only be sent if the agent has the capability `isSupervisor`. They're used by Salesforce to update the Supervisor Call Monitor UI.

For information about publishing events, see `publishEvent()` on page 124.

- SUPERVISOR_CALL_STARTED — The call has started. This event is sent automatically by the base connector for desk phones.
- SUPERVISOR_CALL_CONNECTED — The call has connected. This event is sent automatically on a softphone. It must be sent from a desk phone to connect the call.
- SUPERVISOR_HANGUP — The supervisor has disconnected.
- SUPERVISOR_BARGED_IN— The supervisor has barged in. This event can be called by a hardphone when a supervisor has barged in successfully. To use it, the vendor needs to perform the barge in and then publish this event when the barge in is successful. This event can only be sent if agent has the `hasSupervisorListenIn` capability on page 148.
- CALL_BARGED_IN— This event notifies the agent that a supervisor has barged in. The payload is SupervisedCallInfo on page 170. This event can only be sent if agent has the `hasSupervisorListenIn` capability on page 148.
- MUTE_TOGGLE—Can be used during Supervisor Barge In.
- HOLD_TOGGLE—Can be used during Supervisor Barge In.

- RECORDING_TOGGLE—Can be used during Supervisor Barge In.

SEE ALSO:

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

publishEvent

CHAPTER 17 Service Cloud Voice for Partner Telephony Object Reference

Service Cloud Voice for Partner Telephony requires the use of some Salesforce objects.

> ⊘ **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.

ConversationVendorInfo

This setup object contains information about the Service Cloud Voice telephony vendor system. This object is available in API version 52.0 and later.

Supported Calls

`create()`, `delete()`, `describeSObjects()`, `query()`, `retrieve()`, `update()`, `upsert()`

Special Access Rules

This object requires a license for Service Cloud Voice for Partner Telephony.

Fields

The fields in the ConversationVendorInfo object apply to all three Service Cloud Voice telephony model configurations unless otherwise stated in the field description. For example, if a field applies to just one telephony model setup or is applied differently by telephony model setup, this is stated in the field description.

Field	Details
AwsAccountKey	**Type** string **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** The 12-digit AWS subaccount ID that's automatically provisioned for you when Service Cloud Voice was turned on. Available in API version 55.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Amazon Connect
AwsRootEmail	**Type** email **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** The email address used by Salesforce to create the root user for the provisioned AWS subaccount when Service Cloud Voice was turned on. Available in API version 55.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Amazon Connect
AwsTenantVersion	**Type** double

Field	Details
	Properties Create, Filter, Nillable, Sort, Update **Description** The version number of the SVCTenantStack AWS CloudFormation stack that's deployed. The stack is deployed in AWS region "us-east-1". Available in API version 55.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Amazon Connect
`agentSSOSupported`	**Type** boolean **Properties** Create, Filter, Group, Update **Description** If set to `true`, agents can single sign-on (SSO) into their contact center using Salesforce as the identity provider (IdP). Behind the scenes, Salesforce is used as the SAML IdP in the Single Sign-On connected app for the contact center. If set to `false`, an IdP other than Salesforce is used or an IdP isn't used at all. The default value is `false`. If this value is set to `false` and you want to use Salesforce as the IdP for your contact center, set this value and the `namedCredentialSupported` value to `true` and configure the `service_cloud_voice.PartnerSSO` interface in your Apex integration class. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
`BridgeComponent`	**Type** string **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** The Lightning component used to communicate between the telephony system and other Lightning components. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
`CapabilitiesSupportsQueueManagement`	**Type** boolean **Properties** Create, Filter, Update

Field	Details
	Description If set to `true`, supports queue management. The default value is `false`. Available in API version 56.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
ClientAuthMode	**Type** picklist **Properties** Create, Filter, Group, Restricted picklist, Sort, Update **Description** The client authentication mode. Possible values are: • `Custom` • `Mixed` • `SSO` Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
ConnectorUrl	**Type** url **Properties** Create, Filter, Group, Sort, Update **Description** The URL that hosts your Service Cloud Voice connector. This value could be a Visualforce page or a public URL. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
customConfig	**Type** CustomEntityDefinition **Properties** Create, Filter, Group, Update **Description** The foreign key to the CustomEntityDefinition, which contains partner-specific custom settings. Available in API version 53.0 and later. Applies to the following telephony implementations:

178

Field	Details
	• Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
CustomLoginUrl	**Type** url **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** The URL that hosts your telephony system login page. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
DeveloperName	**Type** string **Properties** Create, Filter, Group, Sort, Update **Description** The unique name of the object in the API. ✏️ Note: Only users with View DeveloperName or View Setup and Configuration permissions can view, group, sort, and filter this field. Applies to the following telephony implementations: • Service Cloud Voice with Amazon Connect • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
IsTaxCompliant	**Type** string **Properties** Create, Defaulted on create, Filter, Group, Sort, Update **Description** Indicates whether the Amazon tax settings for the AWS subaccount provisioned for Service Cloud Voice have been confirmed (`true`). The default value is `false`. Available in API version 55.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Amazon Connect
einsteinConversationInsights Supported	**Type** boolean

Field	Details
	Properties Create, Filter, Group, Update **Description** If set to `true`, Einstein Conversation Insights is turned on. The default value is `false`. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
integrationClass	**Type** ApexClass **Properties** Create, Filter, Group, Update **Description** The foreign key to the partner Apex class implementing supported interfaces. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
IntegrationClassName	**Type** string **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** Deprecated in API version 53.0. Don't set this field. Instead, use `integrationClass`. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
keyProvisioningSupported	**Type** boolean **Properties** Create, Filter, Group, Update **Description** If set to `true`, key provisioning and renewal are automated. The default value is `false`. Available in API version 54.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect

Field	Details
Language	**Type** picklist **Properties** Create, Defaulted on create, Filter, Group, Nillable, Restricted picklist, Sort, Update **Description** The language of the master label (`MasterLabel`) in the user interface. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
MasterLabel	**Type** string **Properties** Create, Filter, Group, Sort, Update **Description** For Service Cloud Voice with Amazon Connect, this field is always set to `Service Cloud Voice`. For Service Cloud Voice with Partner Telephony or Service Cloud Voice with Partner Telephony from Amazon Connect, this field represents the telephony provider display name as it appears in the UI. This label also represents the telephony provider name in the contact center record. Applies to the following telephony implementations: • Service Cloud Voice with Amazon Connect • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
namedCredential	**Type** Named Credential **Properties** Create, Filter, Group, Update **Description** A sample named credential that can be used for Apex callouts to the partner system. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
namedCredentialSupported	**Type** boolean **Properties** Create, Filter, Group, Update

Field	Details
	Description If set to `true` and if supported by the partner telephony, then prescriptive setup through a named credential is enabled allowing server-to-server communication with a named credential. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
NamespacePrefix	**Type** string **Properties** Filter, Group, Nillable, Sort **Description** The namespace prefix that's associated with this object. Applies to the following telephony implementations: • Service Cloud Voice with Amazon Connect • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
partnerContactCenterList Supported	**Type** boolean **Properties** Create, Filter, Group, Update **Description** If set to `true`, enables the customer to select one contact center from a list of multiple contact centers to connect with Salesforce. The default value is `false`. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
partnerPhoneNumbersSupported	**Type** boolean **Properties** Create, Filter, Group, Update **Description** If set to `true`, displays a list of phone numbers used to create contact center channels. The default value is `false`. Available in API version 54.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony

Field	Details
	• Service Cloud Voice with Partner Telephony from Amazon Connect
`partnerTransfer` `DestinationsSupported`	**Type** boolean **Properties** Create, Filter, Group, Update **Description** If set to `true`, allows Salesforce to fetch contact center queues so that Salesforce and contact center queues can be mapped. The default value is `false`. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
`ServerAuthMode`	**Type** picklist **Properties** Create, Filter, Group, Restricted picklist, Sort, Update **Description** Deprecated in API 53.0. Server authentication mode. Set this value to `None`. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
`TelephonySettingsComponent`	**Type** string **Properties** Create, Filter, Group, Nillable, Sort, Update **Description** The name of the Lightning Web Component (LWC) that is used to display additional agent settings in the Omni-Channel widget. This value is in the format `mynamespace:componentName`, where `mynamespace` is the namespace associated with the Service Cloud Voice package that was created, and `componentName` is the FQDN of the Lightning component. Available in API version 54.0 and later. Applies to the following telephony implementation: • Service Cloud Voice with Partner Telephony
`universalCallRecording` `AccessSupported`	**Type** boolean

183

Field	Details
	Properties Create, Filter, Group, Update **Description** If set to `true`, Universal Call Recording Access is turned on. The default value is `false`. If this value is set to `false` and you want to turn on Universal Call Recording, set this value to `true` and configure the service_cloud_voice.RecordingMediaProvider interface in your Apex integration class. Available in API version 54.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
`userSyncingSupported`	**Type** boolean **Properties** Create, Filter, Group, Update **Description** If set to `true`, supports automated user syncing whenever a user is added to or removed from a contact center. The default value is `false`. Available in API version 53.0 and later. Applies to the following telephony implementations: • Service Cloud Voice with Partner Telephony • Service Cloud Voice with Partner Telephony from Amazon Connect
`vendorType`	**Type** picklist **Properties** Create, Filter, Group, Update **Description** Possible values are: • `Amazon_Connect` (For Service Cloud Voice with Amazon Connect) • `BringYourOwnChannelPartner` (For internal use only) • `ServiceCloudVoicePartner` (For Service Cloud Voice with Partner Telephony or Service Cloud Voice with Partner Telephony from Amazon Connect) Available in API version 53.0 and later.

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 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 on page 176. 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 76.

service_cloud_voice.IntelligenceServiceAndSignalsInfo

For guidance on this Apex class, see Send Real-Time Signals on page 76.

service_cloud_voice.IntelligenceServiceProvider

For guidance on this Apex interface, see Send Real-Time Signals on page 76.

service_cloud_voice.IntelligenceServiceRequest

For guidance on this Apex class, see Send Real-Time Signals on page 76.

service_cloud_voice.IntelligenceServiceResponse

For guidance on this Apex class, see Send Real-Time Signals on page 76.

service_cloud_voice.IntelligenceSignalType

For guidance on this Apex class, see Send Real-Time Signals on page 76.

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 agents with Salesforce as an identity provider. See Set Up Single Sign-On on page 38 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 agent 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 agent.

service_cloud_voice.UpdateOrgDomainProvider

Enhanced domains are the current version of My Domain that meets the latest security standards. Some of the 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. When enhanced domain is enabled, the domain value changes and impacts the URLs used by Service Cloud Voice features. Whether the domain changes from enabling enhanced domain or from a manual name change from the setup page, this Apex class notifies you of updated domain information. To learn more about the roll out of the enhanced domain feature, see Enhanced Domains Timeline in Salesforce Help.

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`.

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.

Omni-Channel Doesn't Show Agent Work to Accept or Reject Incoming Call

Issues you could see:

- Omni-Channel widget doesn't show the Agent Work user interface (UI) to accept or reject the call.
- Omni-Channel user interface (UI) goes blank on incoming call.

Possible causes:

- Agent Work creation failed. Check your connector CALL_STARTED event payload and verify that you don't have invalid data. Also, check that the agent is online with the correct Omni-Channel status for the assigned phone channel.
- Salesforce Omni-Channel has an outage, which prevents pushing Agent Work from the server to the agent console UI. Check Omni-Channel service status at trust.salesforce.com. To verify that Agent Work has been created successfully, refresh the page. Omni-Channel should load the Agent Work and allow the agent 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 agent isn't assigned to the required permission set.
- The agent 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 on page 176 record. When you import the Contact Center XML file in Setup, it points to the developer name of the ConversationVendorInfo on page 176.

- 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.