Open CTI Developer Guide

Version 66.0, Spring ’26

Last updated: February 13, 2026

CONTENTS

CHAPTER 1 Get Started with Open CTI

Build and integrate third-party computer-telephony integration (CTI) systems with Salesforce Call Center using a browser-based JavaScript API.

Supported Editions

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience

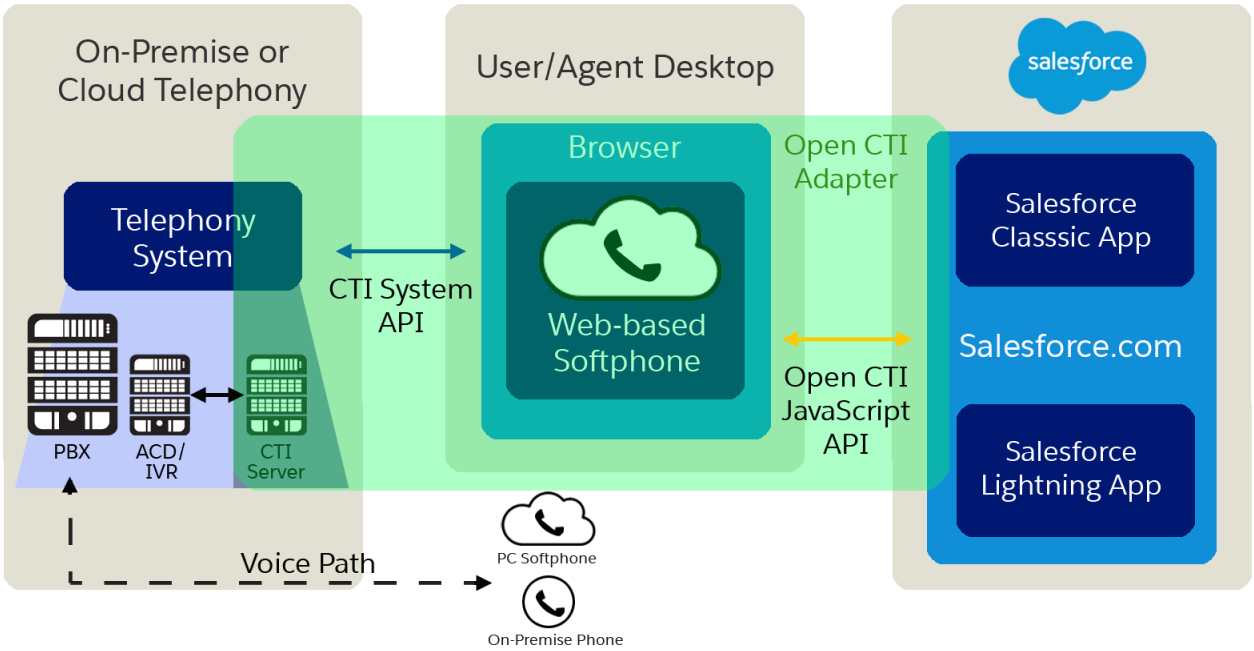
Available in: **Essentials**, **Professional**, **Enterprise**, **Performance**, **Unlimited**, and **Developer** Editions

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

To display CTI functionality in Salesforce, Open CTI uses browsers as clients. With Open CTI, you can make calls from a softphone directly in Salesforce without installing CTI adapters on your machines. After you develop an Open CTI implementation, you can integrate it with Salesforce using Salesforce Call Center.

Note: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience. You can't swap the two Open CTI APIs in custom JavaScript code because they behave and function differently. Make sure that you think about where you want to implement your CTI system before you begin developing.



With Open CTI, you can:

- Build CTI systems that integrate with Salesforce without the use of CTI adapters.
- Create customizable softphones (call-control tools) that function as fully integrated parts of Salesforce and the Salesforce console.
- Provide users with CTI systems that are browser and platform agnostic, for example, CTI for Microsoft® Internet Explorer®, Mozilla® Firefox®, Apple® Safari®, or Google Chrome™ on Mac, Linux, or Windows machines.

To implement Open CTI, it helps if you have a basic familiarity with: CTI, JavaScript, Visualforce, web services, software development, the Salesforce console, and the Salesforce Call Center.

Keep in mind that Open CTI is only available for use with JavaScript pages. The examples in this guide are in JavaScript. You can use Open CTI in JavaScript to embed API calls and processes.

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience.

Method Parity Between Open CTI for Salesforce Classic and Lightning Experience

The methods provided in the two APIs aren't always the same. Some Salesforce Classic methods aren't available in Lightning Experience and some have been renamed.

Open CTI and Other Voice Solutions

Open CTI integrates third-party CTI systems with Salesforce. But do you wonder what came before? Or what the difference is between Open CTI and Sales Dialer?

Customize Open CTI Functionality

Your organization may have complex business processes that are unsupported by Open CTI functionality. Not to worry. When this is the case, the Lightning platform offers advanced administrators and developers several ways to implement custom functionality.

Open CTI Support Policy

Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice.

SEE ALSO:

Salesforce Help: Salesforce Call Center

Salesforce Help: Salesforce Console

Salesforce Help: Supported Browsers

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience.

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

Note: You can't swap the two Open CTI APIs in custom JavaScript code because they behave and function differently. Make sure that you think about where you want to implement your CTI system before you begin developing.

What's the difference between the two Open CTI APIs?

- You connect to the API differently.

  In Salesforce Classic

  ```
  /support/api/66.0/interaction.js
  ```

  In Lightning Experience

  ```
  /support/api/66.0/lightning/opencti_min.js
  ```

- The input syntax for methods is different.

  In Salesforce Classic

  Input example:

  ```
  sampleMethod(var1,var2...)
  ```

In Lightning Experience

Input example:

```
sampleMethod({
 arg1 : value1,
 arg2 : value2,
 ...
})
```

- The two APIs provide similar methods, but a few methods behave differently. The input and output for methods can be different.

Which Open CTI API do I use?

Remember that the APIs can't be swapped. If your users plan to switch between user interfaces, make sure that they understand that the CTI system might behave or function differently depending on what user interface they're working in.

Use Open CTI for Salesforce Classic if...

- You want to make calls using a softphone in Salesforce Classic
- You want to make calls using a softphone in a Salesforce Classic console app

Use Open CTI for Lightning Experience if...

- You want to make calls using a softphone in Lightning Experience
- You want to make calls using a softphone in a Lightning Experience console app

Are there any setup considerations?

To make calls in Lightning Experience, complete the following.

- Create a Lightning app and add the Open CTI Softphone to your utility bar.
- In the call center definition file, the `reqSalesforceCompatibilityMode` item must be set to *`Lightning`* or *`Classic_and_Lightning`*.

Open CTI for Lightning Experience works only in Lightning apps—it doesn't work in Salesforce Classic apps. Even though you can view Salesforce Classic apps in Lightning Experience, those apps are still Classic apps under-the-covers. To check if your app is a Lightning app, use the App Manager in Setup.

If you want your Open CTI implementation to work in Lightning Experience and in a console in Salesforce Classic, develop a unique implementation that uses both Open CTI for Salesforce Classic and Lightning Experience.

SEE ALSO:

Method Parity Between Open CTI for Salesforce Classic and Lightning Experience

Method Parity Between Open CTI for Salesforce Classic and Lightning Experience

The methods provided in the two APIs aren't always the same. Some Salesforce Classic methods aren't available in Lightning Experience and some have been renamed.

Salesforce Classic Method	Available in Lightning Experience?	Notes About Support in Lightning Experience	Go to Salesforce Classic API	Go to Lightning Experience API
`disableClickToDial()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`enableClickToDial()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`getCallCenterSettings()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`getDirectoryNumbers()`	✘	Not yet supported.	Link	N/A
`getPageInfo()`	✔	The same functionality is provided in the Open CTI for Lightning method `getAppViewInfo`.	Link	Link
`getSoftphoneLayout()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`isInConsole()`	✘	Not yet supported.	Link	N/A
`isVisible()`	✔	The same functionality is provided in the Open CTI for Lightning `isSoftphonePanelVisible`.	Link	Link
`notifyInitializationComplete()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`onClickToDial()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`onFocus()`	✔	The same functionality is provided in the Open CTI for Lightning method `onNavigationChange`.	Link	Link
`refreshPage()`	✔	The same functionality is provided in the Open CTI for Lightning method `refreshView`.	Link	Link
`refreshRelatedList()`	✔	The same functionality is provided in the Open CTI for Lightning method `refreshView`.	Link	Link

Salesforce Classic Method	Available in Lightning Experience?	Notes About Support in Lightning Experience	Go to Salesforce Classic API	Go to Lightning Experience API
`reloadFrame()`	✔	The same functionality is provided in the Open CTI for Lightning method `refreshView`.	Link	Link
`runApex()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`saveLog()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`screenPop()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`searchAndGetScreenPopUrl()`	✘	To recreate this functionality, use `searchAndScreenPop` in Open CTI for Lightning	Link	Link
`searchAndScreenPop()`	✔	Uses the same method name in Open CTI for Lightning.	Link	Link
`setSoftphoneHeight()`	✔	The same functionality is provided in the Open CTI for Lightning method `setSoftphonePanelHeight`.	Link	Link
`setSoftphoneWidth()`	✔	The same functionality is provided in the Open CTI for Lightning method `setSoftphonePanelWidth`.	Link	Link
`setVisible()`	✔	The same functionality is provided in the Open CTI for Lightning method `setSoftphonePanelVisibility`.	Link	Link

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Open CTI and Other Voice Solutions

Open CTI integrates third-party CTI systems with Salesforce. But do you wonder what came before? Or what the difference is between Open CTI and Sales Dialer?

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

What came before Open CTI?

Desktop CTI, also known as the CTI Toolkit, is the predecessor to Open CTI. Desktop CTI required adapters to be installed on each call center user's machine. With Open CTI, those user-side adapters are a thing of the past.

Important: Desktop CTI is retired and you must migrate to another solution. Work with your partners to create a Service Cloud Voice implementation.

What about Sales Dialer?

If you're confused between Sales Dialer and Open CTI, don't be. Sales Dialer provides a way to provision numbers and make calls directly from Salesforce. However, if you already have a telephony system in place, Open CTI is the way to go since it integrates to that existing system.

SEE ALSO:

CTI Toolkit Retirement FAQ

Salesforce Help: Guidelines for Making and Receiving Calls

Customize Open CTI Functionality

Your organization may have complex business processes that are unsupported by Open CTI functionality. Not to worry. When this is the case, the Lightning platform offers advanced administrators and developers several ways to implement custom functionality.

Feature	Description
SOAP API	Use standard SOAP API calls when you want to add functionality to a composite application that processes only one type of record at a time and does not require any transactional control (such as setting a Savepoint or rolling back changes). For more information, see the SOAP API Developer Guide.
Visualforce	Visualforce consists of a tag-based markup language that gives developers a more powerful way of building applications and customizing the Salesforce user interface. With Visualforce you can: • Build wizards and other multistep processes. • Create your own custom flow control through an application. • Define navigation patterns and data-specific rules for optimal, efficient application interaction. For more information, see the Visualforce Developer's Guide.

Feature	Description
Console API	The Salesforce Console Integration Toolkit and the Lightning Console JavaScript APIs let you implement custom functionality for the Salesforce console. For example, you can use the Console API to display Visualforce pages or third-party content as tabs in the Salesforce console. For more information, see the Salesforce Console Developer Guide.
Apex	Use Apex if you want to: • Create Web services. • Create email services. • Perform complex validation over multiple objects. • Create complex business processes that aren't supported by Flow Builder. • Create custom transactional logic (logic that occurs over the entire transaction, not just with a single record or object). • Attach custom logic to another operation, such as saving a record, so that it occurs whenever the operation is executed, regardless of whether it originates in the user interface, a Visualforce page, or from SOAP API. For more information, see the Apex Developer Guide.

Open CTI Support Policy

Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

For more information, see this Knowledge Article.

Backward Compatibility

Salesforce strives to make backward compatibility easy when using Open CTI.

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

Each new Salesforce release consists of two components:

- A new release of platform software that resides on Salesforce systems
- A new version of the API

Open CTI matches the API version for any given release. For example, if the current version of SOAP API is 66.0, then there's also a version 66.0 of Open CTI.

We maintain support for each Open CTI version across releases of the platform. Open CTI is backward compatible in that an application created to work with a given Open CTI version will continue to work with that same Open CTI version in future platform releases.

Salesforce doesn't guarantee that an application written against one Open CTI version will work with future Open CTI versions: Changes in method signatures and data representations are often required as we continue to enhance Open CTI. However, we strive to keep Open CTI consistent from version to version with minimal changes required to port applications to newer Open CTI versions.

For example, an application written using Open CTI version 37.0, which shipped with the Summer '16 release, will continue to work with Open CTI version 37.0 on the Winter '17 release and on future releases. However, that same application might not work with Open CTI version 38.0 without modifications to the application.

API Support

Salesforce is committed to supporting each Open CTI version for a minimum of three years from the date of its first release.

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

To improve the quality and performance of Open CTI, versions that are more than three years old might not be supported.

When a Open CTI version is scheduled to be unsupported, a no-longer-available notice will be given at least one year before support for the version ends. Salesforce will directly notify customers using Open CTI versions that will no longer be available.

CHAPTER 2 Call Center Definition Files

A call center definition file specifies a set of fields and values that are used to define a call center in Salesforce for a particular softphone. Salesforce uses call center definition files to support the integration of Salesforce CRM Call Center with multiple CTI system vendors.

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

A call center in Salesforce CRM Call Center must have a call center definition file that works specifically with a softphone. If you build a custom softphone with Open CTI, you must write a call center definition file to support it. The first instance of a call center for a particular softphone must be defined by importing the adapter's call center definition file into Salesforce. Subsequent call centers can be created by cloning the original call center that was created with the import.

If your organization modifies a softphone or builds a new one, you must customize the softphone's call center definition file so that it includes any additional call center information that is required. For example, if you are building a softphone for a system that supports a backup server, your call center definition file should include fields for the backup server's IP address and port number. Softphones for systems that don't have a backup server, don't need those fields in their associated call center definition files.

Use a text or XML editor to define a call center definition file.

Important: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience. The `reqSalesforceCompatibilityMode` item in your call center definition file identifies the user interface you plan to use—Salesforce Classic, Lightning Experience, or both. If no value is specified, the default is *`Classic`*. This item is optional, but to make calls in Lightning Experience you must specify *`Lightning`* or *`Classic_and_Lightning`*.

Sample Call Center Definition File

Each call center definition file looks different. This example shows you what a call center definition file looks like for an org using Salesforce Classic and Lightning Experience.

SEE ALSO:

Salesforce Help: Set Up a Call Center

Salesforce Help: Creating a Call Center

Call Center Definition File Format

A call center definition file consists of three XML elements: `callCenter`, `section`, and `item`.

This list provides details about the properties and attributes of each element:

- `callCenter`

  This element represents a definition for a single call center phone system. At least one `<callCenter>` element must be included in every call center definition file. A `<callCenter>` element consists of one or more `<section>` elements.

- `section`

  This element represents a grouping of related data fields, such as server information or dialing prefixes. When a call center is edited in Salesforce, fields are organized by the section to which they are assigned. A `<section>` element belongs to a single `<callCenter>` element, and consists of one or more `<item>` elements.

  Attributes:

Name	Type	Required?	Description
`sortOrder`	Positive Integer	Required	The order in which the section appears when the call center is edited in Salesforce. For example, a section with `sortOrder="1"` comes just before a section with `sortOrder="2"`. The values for `sortOrder` must be non-negative integers, and no numbers can be skipped within a single call center definition. For example, if there are three section elements in a call center definition file, one `<section>` element must have `sortOrder="0"`, one `<section>` element must have `sortOrder="1"`, and one `<section>` element must have `sortOrder="2"`.
`name`	String	Required	The internal name of the section as defined in the Salesforce database. You can use this value to refer to the section when writing custom adapter or SoftPhone code. Names must be composed of only alphanumeric characters with no white space or other punctuation. They are limited to 40 characters each. Names beginning with `req` are reserved for required Salesforce sections only. Other reserved words that cannot be used for the `name` attribute include `label`, `sortOrder`, `internalNameLabel`, and `displayNameLabel`.

Name	Type	Required?	Description
`label`	String	Optional	The name of the section when viewed in Salesforce. Labels can be composed of any string of UTF-8 characters. They are limited to 1000 characters each.

- `item`

  This element represents a single field in a call center definition, such as the IP address of a primary server or the dialing prefix for international calls. When call centers are edited in Salesforce, each `<item>` element is listed under the section to which it belongs. You can have multiple `<item>` elements in a `<section>` element.

  Attributes:

Name	Type	Required?	Description
`sortOrder`	Positive Integer	Required	The order in which the item appears when the call center is edited in Salesforce. For example, an item with `sortOrder="1"` comes just before an item with `sortOrder="2"`. The values for `sortOrder` must be non-negative integers, and no numbers can be skipped within a single call center definition. For example, if there are three item elements in a call center definition file, one `<item>` element must have `sortOrder="0"`, one `<item>` element must have `sortOrder="1"`, and one `<item>` element must have `sortOrder="2"`.
`name`	String	Required	The internal name of the item as defined in the Salesforce database. You can use this value to refer to the item when writing a custom adapter or SoftPhone code. Names must be composed of only alphanumeric characters with no white space or other punctuation. They are limited to 40 characters each. Names beginning with `req` are reserved for required Salesforce sections only. Other reserved words that cannot be used for the `name` attribute include `label`, `sortOrder`, `internalNameLabel`, and `displayNameLabel`.
`label`	String	Optional	The name of the item when viewed in Salesforce. Labels can be composed of any string of UTF-8 characters. They are limited to 1,000 characters each.

SEE ALSO:

Salesforce Help: Call Center Definition Files

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Required Call Center Elements and Attributes

The call center definition file must include the required `<item>` elements in the `<section>` element.

Important: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience. The `reqSalesforceCompatibilityMode` item in your call center definition file identifies the user interface you plan to use—Salesforce Classic, Lightning Experience, or both. If no value is specified, the default is *`Classic`*. This item is optional, but to make calls in Lightning Experience you must specify *`Lightning`* or *`Classic_and_Lightning`*.

<item> Name	Description	Supported in Salesforce Classic	Supported in Lightning Experience

`reqAdapterUrl`	Represents the location of where the CTI adapter or softphone is hosted. For example:
	`http://localhost:11000`
	Relative URLs are allowed for Visualforce pages. For example:
	`: /apex/softphone`
	If you add Canvas applications to Open CTI, those apps can trump `reqAdapterUrl` when specified.
	Note: To implement in a Lightning Experience org, use *`https`* in your URL.
		✓	✓

`reqCanvasApiName`	Represents the API name associated with any Canvas applications added to your call center. Required if you add canvas apps to Open CTI.	✓	✗ Not supported
`reqCanvasNamespace`	Represents the namespace associated with any Canvas applications added to your call center. Required if you add canvas apps to Open CTI.	✓	✗ Not supported
`reqDisplayName`	Represents the name of the call center as displayed in Salesforce. It must have a `sortOrder` value of `1`. A value for `reqDisplayName` has a maximum length of 1,000 UTF-8 characters.	✓	✓
`reqInternalName`	Represents the unique identifier for the call center in the database. It must have a `sortOrder` value of `0`, and its value must be specified in the call center definition. A value for `reqInternalName` must be composed of no more than 40 alphanumeric characters with no white space or other punctuation. It must start with an alphabetic character and must be unique from the `reqInternalName` of all other call centers defined in your organization.	✓	✓

<item> Name	Description	Supported in Salesforce Classic	Supported in Lightning Experience
`reqSoftphoneHeight`	Represents the height of the softphone in pixels as displayed in Salesforce. Note: If you're using Open CTI for Lightning Experience, enter a number from 240 through 2,560. Value is in pixels (*px*).	✓	✓
`reqSoftphoneWidth`	Represents the width of the softphone in pixels as displayed in Salesforce. Note: If you're using Open CTI for Lightning Experience, enter a number from 200 through 1,920. Value is in pixels (*px*).	✓	✓
`reqUseApi`	Represents that the call center is using Open CTI (`true`) or not (`false`).	✓	✓

If needed, you can add more `<item>` elements to this section.

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Optional Call Center Elements and Attributes

The call center definition file can include optional `<item>` elements in the `<section>` element.

In addition to the required elements, you can add optional elements to configure a softphone.

Important: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience. The `reqSalesforceCompatibilityMode` item in your call center definition file identifies the user interface you plan to use—Salesforce Classic, Lightning Experience, or both. If no value is specified, the default is *`Classic`*. This item is optional, but to make calls in Lightning Experience you must specify *`Lightning`* or *`Classic_and_Lightning`*.

<item> Name	Description	Supported in Salesforce Classic	Supported in Lightning Experience
`reqSalesforceCompatibilityMode`	Determines where the softphone is visible. If no value is specified, the default is `Classic`. Note: • To display the softphone in Lightning Experience, specify *`Lightning`*.	✓	✓

<item> Name	Description	Supported in Salesforce Classic	Supported in Lightning Experience
	• To display the softphone in Salesforce Classic, specify *Classic*. • To display the softphone in both user interfaces, specify *Classic_and_Lightning*.		
reqStandbyUrl	Represents the location that hosts the secondary softphone. The standby softphone is used after the timeout period for the primary softphone has elapsed and the notifyInitializationComplete() for Salesforce Classic for Salesforce Classic method hasn't been called within the required timeout period. When you specify a standby URL, you must also specify the reqTimeout field.	✔	✔
reqTimeout	Represents the time in milliseconds after which the standby URL is used to host the softphone. Before the timeout period has elapsed, the softphone displays a loading icon indicating that the softphone is initializing. When you specify a required timeout, you must also specify the reqStandbyUrl field.	✔	✔

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Specify Values for <item> Elements

With the exception of the reqInternalName <item>, whose value must always be specified in a call center definition file, you can specify <item> values either in the call center definition file or in Salesforce once the definition file has been imported.

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

To specify a value for an `<item>` element in a call center definition file, place the value between the opening and closing tags of the `<item>`. For example:

```
<item sortOrder="0" name="reqInternalName" label="Call Center Internal
Label">MyCallCenter</item>
```

sets the value of the `reqInternalName` `<item>` to `MyCallCenter`. Note that any `<item>` value other than the value for `reqInternalName` can be edited in Salesforce after the call center definition is imported.

Sample Call Center Definition File

Each call center definition file looks different. This example shows you what a call center definition file looks like for an org using Salesforce Classic and Lightning Experience.

Important: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience. The `reqSalesforceCompatibilityMode` item in your call center definition file identifies the user interface you plan to use—Salesforce Classic, Lightning Experience, or both. If no value is specified, the default is *`Classic`*. This item is optional, but to make calls in Lightning Experience you must specify *`Lightning`* or *`Classic_and_Lightning`*.

Sample XML for an Org Using Salesforce Classic

```
<!--
     All sections and items whose name value begins with "req" are
     required in a valid call center definition file. The sortOrder
     and label attributes can be changed for all required sections
     and items except reqGeneralInfo, reqInternalName, and
     reqDisplayName, in which only the label attribute can be altered.

     Note that the value for the reqInternalName item is limited to
     40 alphanumeric characters and must start with an alphabetic
     character. reqInternalName must be unique for all call centers
     that you define.
-->
<callCenter>
   <section sortOrder="0" name="reqGeneralInfo" label="General Information">
    <item sortOrder="0" name="reqInternalName" label="InternalName">DemoAdapter</item>
    <item sortOrder="1" name="reqDisplayName" label="Display Name">Demo Call Center
Adapter</item>
    <item sortOrder="2" name="reqAdapterUrl" label="CTI Adapter
URL">https://domain:port/softphone</item>
    <item sortOrder="3" name="reqUseApi" label="Use CTI API">true</item>
    <item sortOrder="4" name="reqSoftphoneHeight" label="Softphone Height">300</item>
    <item sortOrder="5" name="reqSoftphoneWidth" label="Softphone Width">500</item>
    <item sortOrder="6" name="reqSalesforceCompatibilityMode" label=" Salesforce
Compatibility Mode">Classic</item>
   </section>
   <section sortOrder="1" name="reqDialingOptions" label="Dialing Options">
    <item sortOrder="0" name="reqOutsidePrefix" label="Outside Prefix">9</item>
    <item sortOrder="1" name="reqLongDistPrefix" label="Long Distance Prefix">1</item>
   <item sortOrder="2" name="reqInternationalPrefix" label="International Prefix">01</item>
```

```
   </section>
</callCenter>
```

Sample XML for an Org Using Lightning Experience

```
<callCenter>
   <section sortOrder="0" name="reqGeneralInfo" label="General Information">
    <item sortOrder="0" name="reqInternalName" label="InternalName">OpenCTI</item>
    <item sortOrder="1" name="reqDisplayName" label="Display Name">OpenCTI</item>
    <item sortOrder="2" name="reqAdapterUrl" label="CTI Adapter
URL">https://domain:port/softphone</item>
    <item sortOrder="3" name="reqUseApi" label="Use CTI API">true</item>
    <item sortOrder="4" name="reqSoftphoneHeight" label="Softphone Height">300</item>
    <item sortOrder="5" name="reqSoftphoneWidth" label="Softphone Width">500</item>
   <item sortOrder="6" name="reqSalesforceCompatibilityMode" label="Salesforce Compatibility
 Mode">Lightning</item>
   </section>
   <section sortOrder="1" name="reqDialingOptions" label="Dialing Options">
    <item sortOrder="0" name="reqOutsidePrefix" label="Outside Prefix">9</item>
    <item sortOrder="1" name="reqLongDistPrefix" label="Long Distance Prefix">1</item>
   <item sortOrder="2" name="reqInternationalPrefix" label="International Prefix">01</item>

   </section>
</callCenter>
```

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

CHAPTER 3 Working with Open CTI

You can use Open CTI to increase agent efficiency, configure your softphone, and complete many more tasks.

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

With Open CTI, you can:

- Set the height or width of a softphone
- Enable or disable click-to-dial
- Return a call center definition file's settings
- Determine if a user is in the Salesforce console
- Show or hide a softphone in the Salesforce console
- Return information about a page
- Execute an Apex method from an Apex class that's exposed in Salesforce
- Save or update an object in Salesforce
- Search keywords in Salesforce and screen pop any matching records as defined in a softphone layout

Before developing an Open CTI implementation, learn how to connect to Open CTI and review the best practices.

Connect to Open CTI

The first portion of any JavaScript code that uses the Open CTI must make the toolkit available to the JavaScript code. The syntax for this is different depending on whether you are embedding JavaScript in a Visualforce page or a page developed using any web technologies and served from a third-party domain.

Open CTI Demo Adapter

We've put together a demo adapter package that lets you test drive Open CTI for Lightning Experience. The package provides a demo softphone that highlights and demonstrates the main features of Open CTI for Lightning Experience without even connecting to a phone system.

Open CTI and Security

We recommend that all Open CTI implementations use HTTPS in the `reqAdapterUrl` element in their call center definition file. Using HTTPS ensures that traffic between your telephony server and Salesforce is encrypted.

Connect to Open CTI

The first portion of any JavaScript code that uses the Open CTI must make the toolkit available to the JavaScript code. The syntax for this is different depending on whether you are embedding JavaScript in a Visualforce page or a page developed using any web technologies and served from a third-party domain.

Tip: The version of Open CTI is in the URL.

For Visualforce Pages

For Visualforce pages or any source other than a custom `onclick` JavaScript button, specify a `<script>` tag that points to the Open CTI JavaScript library file.

In Salesforce Classic:

```
<apex:page>
        <script src="/support/api/66.0/interaction.js" type="text/javascript"></script>

     ...
</apex:page>
```

In Lightning Experience:

```
<apex:page>
         <script src="/support/api/66.0/lightning/opencti_min.js"
type="text/javascript"></script>
     ...
</apex:page>
```

For Visualforce, we recommend using a relative path to include `interaction.js` or `opencti_min.js`.

For a Third-Party Domain

For third-party domains, specify an absolute URL to `interaction.js` or `opencti_min.js` to use the toolkit.

In Salesforce Classic:

```
<script
src="https://MyDomainName--PackageName.vf.force.com/support/api/66.0/interaction.js"
type="text/javascript"></script>
```

In Lightning Experience:

```
<script
src="https://MyDomainName--PackageName.vf.force.com/support/api/66.0/lightning/opencti_min.js"
 type="text/javascript"></script>
```

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Salesforce Help: My Domain URL Formats

Open CTI Demo Adapter

We've put together a demo adapter package that lets you test drive Open CTI for Lightning Experience. The package provides a demo softphone that highlights and demonstrates the main features of Open CTI for Lightning Experience without even connecting to a phone system.

To learn more about the demo adapter, go to Lightning Open CTI. You can also check out the demo adapter code on GitHub.

Note: The Open CTI demo adapter is an archived repository. Salesforce do not provide feature support or add new features for the demo adapter, and there will be limited support if you are using the Open CTI demo adapter for your Open CTI implementations. If you clone the Open CTI demo adapter archived repository, you must also download the latest `ant-salesforce.jar` and replace the pre-existing one in the repository to avoid any errors with the latest Salesforce APIs. See Ant Migration Tool Guide.

Open CTI and Security

We recommend that all Open CTI implementations use HTTPS in the `reqAdapterUrl` element in their call center definition file. Using HTTPS ensures that traffic between your telephony server and Salesforce is encrypted.

By using HTTPS, Open CTI inherits all the benefits of browser and cloud-based security because it points directly to the softphone provider using a secure connection. When users access the softphone on any Salesforce page, it's displayed in an iFrame and all messages from the softphone to Salesforce are sent via Window.postMessage() methods. For each message sent, the target is restricted to Salesforce. For each message received, Salesforce verifies its format and the sender origin.

Tip: For Salesforce Classic console apps, if your CTI phone is running on a server with a non-standard port, make sure to include the port number in your domain. For example, if your server is called `myserver` and your port number is `8500`, include `myserver:8500` as an allowed URL in your Salesforce console. This setting doesn't apply for Lightning console apps.

SEE ALSO:

Required Call Center Elements and Attributes

Mozilla Developer Network: Window.postMessage() method

Salesforce Help: Whitelist Domains for a Salesforce Console in Salesforce Classic

Asynchronous Calls with

Open CTI lets you issue asynchronous calls. Asynchronous calls allow the client-side process to continue instead of waiting for a callback from the server.

To issue an asynchronous call, you must include an extra parameter with the API call, referred to as a callback function. Once the result is ready, the server invokes the callback method with the result.

Asynchronous syntax:

In Salesforce Classic:

```
method('arg1','arg2', ..., callback_method);
```

In Lightning Experience:

```
method({callback : function})
```

Example:

In Salesforce Classic:

```
//Set softphone height
   sforce.interaction.cti.setSoftphoneHeight(300, callback);
```

In Lightning Experience:

```
//Disable clickToDial
   sforce.opencti.disableClickToDial({callback: callback});
```

Note: The call result depends on the execution context. For example, calling `setSoftphoneWidth()` in the standard Salesforce application has no effect, but calling `setSoftphoneWidth()` in the Salesforce console resizes the width of the softphone.

Sample HTML Page Using Open CTI Using Open CTI

Each implementation of Open CTI can look different. This example shows you how to add CTI functionality to the Salesforce user interface using an HTML page.

This example assumes that you've already imported a call center definition file into your Salesforce org. The sample HTML page can be stored on Salesforce as a Visualforce page or on a third-party domain.

1. Create an HTML page.

2. Cut and paste the following sample code into your HTML page.

   This code demonstrates various functions of Open CTI.

   Note: Keep in mind that to make calls in Lightning Experience, you must first create a Lightning app and add the Open CTI Softphone utility.

   Sample Code for Salesforce Classic

```
<html>
<head>
      <!-- Imports Open CTI JavaScript library. Point to a valid Salesforce domain.
-->
      <script src="https://domain:port/support/api/66.0/interaction.js"></script>
      <script type="text/javascript">
             // Callback of API method: isInConsole
             var isInConsoleCallback = function (response) {
                  // Returns true if method is executed in Salesforce console, false
 otherwise.
                  if (response.result) {
                      alert('Softphone is in Salesforce console.');
                  }
                  else {
                      alert('Softphone is not in Salesforce console.');
                  }
              };
              // Invokes API method: isInConsole
              function isInConsole() {
                       sforce.interaction.isInConsole(isInConsoleCallback);
              }
              // Callback of API method: getCallCenterSettings
              var getCallCenterSettingsCallback = function (response) {
                     // Result returns call center settings as a JSON string.
                     if (response.result) {
                            alert(response.result);
                     }
                     else {
                            alert('Error retrieving call center settings ' +
response.error);
                     }
              };
              // Invokes API method: getCallCenterSettings
              function getCallCenterSettings() {

sforce.interaction.cti.getCallCenterSettings(getCallCenterSettingsCallback);
              }
              // Callback of API method: setSoftphoneHeight
              var setSoftphoneHeightCallback = function (response) {
                     // Returns true if SoftPhone height was set successfully, false
 otherwise.
                      if (response.result) {
                         alert('Setting softphone height to 300px was successful.');

                      }
                      else {
                         alert('Setting softphone height failed.');
```

```
                    }
                };
                // Invokes setSoftphoneHeight API method.
                function setSoftphoneHeight() {
                        sforce.interaction.cti.setSoftphoneHeight(300,
setSoftphoneHeightCallback);
                }
                // Callback of API method: getPageInfo
                var getPageInfoCallback = function (response) {
                       if (response.result) {
                              alert(response.result);
                       }
                       else {
                              alert('Error occured while trying to get page info: ' +
 response.error);
                       }
                }
                // Invokes API method getPageInfo
                function getPageInfo() {
                        sforce.interaction.getPageInfo(getPageInfoCallback);
                }
      </script>
</head>
<body>
      <button onclick="isInConsole();">isInConsole</button><br/>
      <button onclick="getCallCenterSettings();">getCallCenterSettings</button><br/>

      <button onclick="setSoftphoneHeight();">setSoftphoneHeight(300)</button><br/>
      <button onclick="getPageInfo();">getPageInfo</button>
</body>
</html>
```

Sample Code for Lightning Experience

```
<apex:page >
  <!-- Begin Default Content -->
  <h1>Congratulations!</h1>
  This is your sample page.
  <!-- End Default Content -->
<html>
<head>
   <!-- Imports Open CTI JavaScript library. Point to a valid Salesforce domain.
-->
   <script
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
      // Callback of API method: setSoftphonePanelHeight
      var setSoftphonePanelHeightCallback = function(response) {
          // Returns true if setSoftphonePanelHeight method is executed successfully,
 false otherwise
            if (response.result) {
               alert('setSoftphonePanelHeight is successfully executed.');
            }
            else {
               alert('setSoftphonePanelHeight failed.);
```

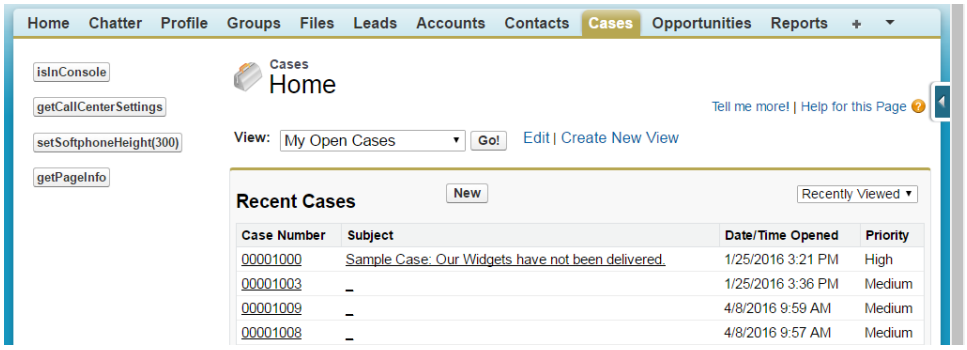
```
            }
      };
      // Invokes API method: setSoftphonePanelHeight
      function setSoftphonePanelHeight() {
         sforce.opencti.setSoftphonePanelHeight({
            heightPX: 500,
            callback: setSoftphonePanelHeightCallback
         });
      }
      // Callback of API method: setSoftphonePanelWidth
      var setSoftphonePanelWidthCallback = function(response) {
           // Returns true if setSoftphonePanelWidth method is executed successfully,
false otherwise
            if (response.result) {
               alert('setSoftphonePanelWidth is successfully executed.');
            }
            else {
               alert('setSoftphonePanelWidth failed.');
            }
      };
      // Invokes API method: setSoftphonePanelWidth
      function setSoftphonePanelWidth() {
         sforce.opencti.setSoftphonePanelWidth({
            widthPX: 500,
            callback: setSoftphonePanelHeightCallback
         });
      }
      // Callback of API method: setSoftphoneItemIcon
      var setSoftphoneItemIconCallback = function(response) {
            // Returns true if setSoftphoneItemIcon method is executed successfully,
false otherwise
            if (response.result) {
               alert('setSoftphoneItemIcon is successfully executed.');
            }
            else {
               alert('setSoftphoneItemIcon failed.');
            }
      };
      // Invokes API method: setSoftphoneItemIcon
      function setSoftphoneItemIcon() {
         sforce.opencti.setSoftphoneItemIcon({
            key: 'call',
            callback: setSoftphoneItemIconCallback
         });
      }
      // Callback of API method: setSoftphoneItemLabel
      var setSoftphoneItemLabelCallback = function(response) {
           // Returns true if setSoftphoneItemLabel method is executed successfully,
false otherwise
            if (response.result) {
               alert('setSoftphoneItemLabel is successfully executed.');
            }
            else {
               alert('setSoftphoneItemLabel failed.');
```

```
            }
        };
        // Invokes API method: setSoftphoneItemLabel
        function setSoftphoneItemLabel() {
            sforce.opencti.setSoftphoneItemLabel({
                Label: 'MySoftphone',
                callback: setSoftphoneItemLabelCallback
            });
        }
    </script>
</head>
<body>
    <button
onclick="setSoftphonePanelHeight();">setSoftphonePanelHeight({heightPX:500})</button><br/>

    <button
onclick="setSoftphonePanelWidth();">setSoftphonePanelWidth({widthPX:500})</button><br/>

    <button
onclick="setSoftphoneItemIcon();">setSoftphoneItemIcon({key:'call'})</button><br/>
    <button
onclick="setSoftphoneItemLabel();">setSoftphoneItemLabel({label:'MySoftphone'})</button>
</body>
</html>
</apex:page>
```

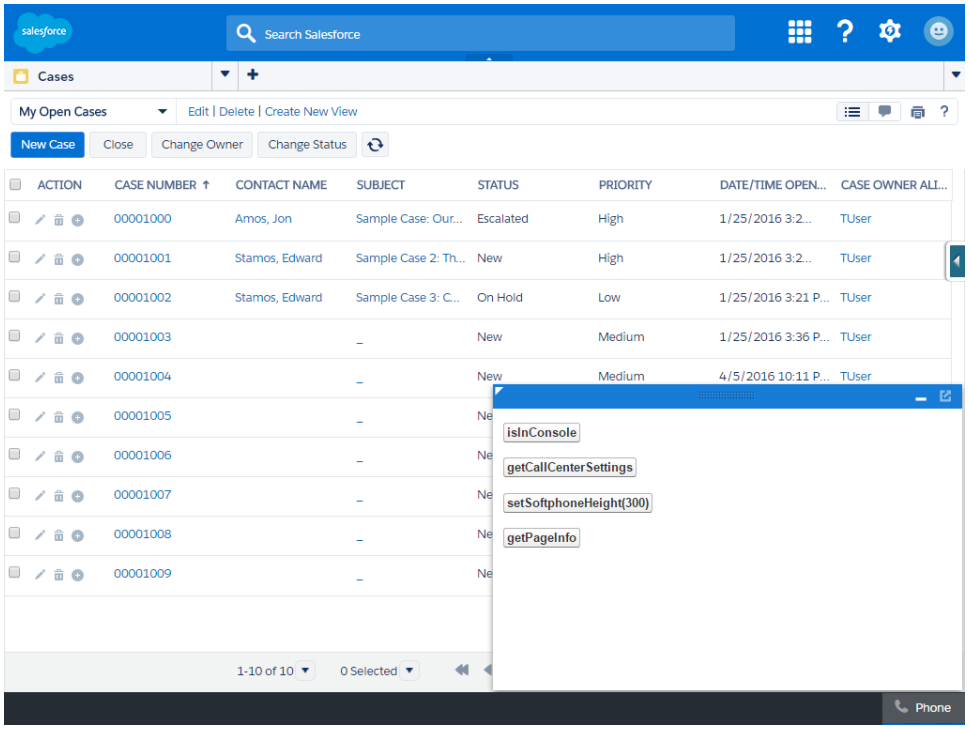
After you create the HTML page, add the URL to the call center definition file. The softphone is rendered differently depending on your user interface:

- In Salesforce Classic, on the left of the page
- In the Salesforce Classic Console App, in a footer
- In Lightning Experience, in a footer

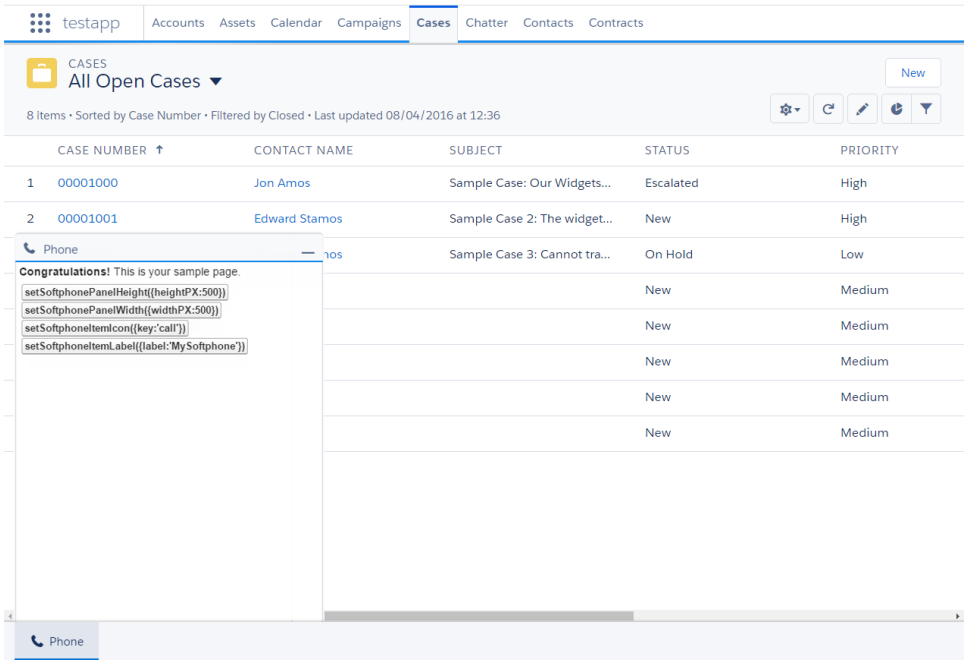
Output of Sample HTML Page in Salesforce Classic



Output of Sample HTML Page in a Salesforce Classic Console App



Output of Sample HTML Page in a Lightning Experience App



SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Sample Code for Using Lightning Message Service with Open CTI

You can use the Lightning Message Service API to communicate with an Open CTI softphone. This example displays three buttons that subscribe, publish, and remove a message channel `servicedev1_SampleMC__c`.

To use Open CTI, you must reference either `opencti.js` or `opencti_min.js` version 47.0 on your softphone adapter page. If you don't have these files, your Salesforce representative can provide them.

In the `<head>` of the HTML file, include the filepath for `opencti_min.js`.

First, define a variable for the name of your message channel in the JavaScript section of your code. Here, we assign the message channel string `SAMPLEMC_c` to the variable `SAMPLEMC`.

Next, create the variable `SAMPLEMC_SUBSCRIPTION` to hold the subscription object and assign it a null value.

The `subscribeToSampleMC()` method first checks whether the subscription variable is empty. If the method is empty, it calls the `sforce.opencti.subscribe()` method.

The `sforce.opencti.subscribe()` method has three parameters:

`channelName`
The name of the message channel.

`listener`
The listener function that's invoked when a message is sent on the message channel.

`callback`
The function that receives the subscription object. Assign this value to a variable such as `SAMPLEMC_SUBSCRIPTION`.

The `subscribeToSampleMC()` method sets the `<span>` tag in the HTML to *`true`*.

Similarly, the `unsubscribeToSampleMC()` method checks whether there is a subscription object. If so, it calls `sforce.opencti.unsubscribe()`. The method `sforce.opencti.unsubscribe()` has two parameters:

subscription
The subscription object that you want to unsubscribe from.

callback
The function that handles the success or error output.

The `unsubscribeToSampleMC()` method sets the `<span>` tag in the HTML to *`false`*.

The `publishSampleMC()` method is passed the message content that we want to publish. A message is a serialized JSON object. You can pass data such as strings, numbers, objects, and Boolean values. A message cannot contain functions and symbols.

In this example, the message contains three strings:

- Name of the sender, "LightningMessageService_OpenCTI_TestPage"
- Type of message, "SampleMC"
- Time when the message was sent

The `publishSampleMC()` method calls `sforce.opencti.publish()` method, which takes three parameters:

channelName
The name of the message channel to publish to.

message
The message to publish as a serialized JSON object.

callback
The function that handles the success or error output.

The HTML markup displays the message channel's name, indicates whether the channel is subscribed, and shows buttons to subscribe, unsubscribe, and publish on the channel. When clicked, these buttons call the respective `subscribeToSampleMC()`, `unsubscribeToSampleMC()`, and `publishSampleMC()` functions. The text area with the `id` set to `opencti_testMessageTextArea` shows the message text.

Example:

```
<html>
 <head>
 <script type="text/javascript"
src="https://domain:port/support/api/47.0/lightning/opencti_min.js"></script>
 <script type="text/javascript">
  <script type="text/javascript">
  var SAMPLEMC = "SAMPLEMC__c";
  var SAMPLEMC_SUBSCRIPTION = null;

  function subscribeToSampleMC() {
   if (!SAMPLEMC_SUBSCRIPTION) {
    sforce.opencti.subscribe({channelName: SAMPLEMC, listener: onPublishMessage,
callback: subscribeSampleMCCallback});
    let mcSubscribedToggle = document.querySelector("#opencti_mcSubscribedToggle");
    mcSubscribedToggle.innerHTML = "true";
   }
  }

  function unsubscribeToSampleMC() {
   if (SAMPLEMC_SUBSCRIPTION) {
    sforce.opencti.unsubscribe({subscription: SAMPLEMC_SUBSCRIPTION, callback:
lightningMessageServiceCallback});
    let mcSubscribedToggle = document.querySelector("#opencti_mcSubscribedToggle");
    mcSubscribedToggle.innerHTML = "false";
    SAMPLEMC_SUBSCRIPTION = null;
   }
  }

  function publishSampleMC() {
   const message = {
    from: "LightningMessageService_OpenCTI_TestPage",
    type: "SampleMC",
    time: new Date().toLocaleTimeString()
   };

   sforce.opencti.publish({channelName: SAMPLEMC, message: message, callback:
lightningMessageServiceCallback});
  }

  function lightningMessageServiceCallback(result) {
   if (result.success) {
    console.log(result.returnValue);
   } else {
    console.log(result.errors);
   }
  }

  function subscribeSampleMCCallback(result) {
```

```
    if (result.success) {
     SAMPLEMC_SUBSCRIPTION = result.subscription;
    } else {
     console.log(result.errors);
    }
   }

   function onPublishMessage(message) {
    var textArea = document.querySelector("#opencti_testMessageTextArea");
    textArea.innerHTML = message ? JSON.stringify(message, null, '\t') : 'no message
payload';
   }
 </script>
 </head>
 <body>
  <div>
   <p>MessageChannel: SampleMC</p>
   <p>Subscribed: <span id="opencti_mcSubscribedToggle">false</span></p>
   <br/>
   <input value="Subscribe" type="button" onclick="subscribeToSampleMC()"/>
   <input value="Unsubscribe" type="button" onclick="unsubscribeToSampleMC()"/>
   <input value="Publish" type="button" onclick="publishSampleMC()"/>
   <br/>
   <p>Received message:</p>
   <textarea id="opencti_testMessageTextArea" class="opencti_testMessageTextArea"
rows="10" style="disabled:true;resize:none;width:100%;"/>
  </div>
 </body>
</html>
```

Use Message Channels Created Within Your Org

Here's an example that shows how to use a Lightning Message Channel developed within your org.

```
<apex:page>
    <script>
        var SAMPLEMC = "SAMPLEMC__c";
    </script>
</apex:page>
```

Here, a custom Lightning Message Channel is referenced by its name `SAMPLEMC__c`. The syntax refers to a custom instance of the LightningMessageChannel metadata type. The `__c` suffix indicates that it's a custom, but note that it is not a custom object. For more information, see Create a Message Channel.

If your org has a namespace, don't include it in your message channel name. For example, if your org's namespace is *MyNamespace*, then the message channel name remains `SAMPLEMC__c`.

Use Message Channels Created Outside Your Org

To use Lightning Message Channels from packages created by developers outside of your org, reference them with the syntax `Namespace_name__c`. For example, if `SAMPLEMC` was not local to your org and came from a package with the namespace `SamplePackageNamespace`, the syntax would be `SamplePackageNamespace_SAMPLEMC_c`.

SEE ALSO:

- Lightning Message Service Methods for Lightning Experience
- *Lightning Aura Components Developer Guide: Communicating Across the DOM with Lightning Message Service*
- *Visualforce Developer Guide: Communicating Across the DOM with Lightning Message Service*

Work with Canvas

To integrate Open CTI with external applications that require authentication methods, such as signed requests or OAuth 2.0 protocols, Salesforce recommends that you use Canvas.

Important: Open CTI for Lightning Experience doesn't support Canvas.

Canvas and Open CTI are similar—they're a set of tools and JavaScript APIs that developers can use to add third-party systems to Salesforce. However, one of the benefits of Canvas, is the ability to choose authentication methods.

Note: For a canvas app to appear in a Salesforce console, you must add it to the console as a custom console component. For more information, see the *Canvas Developer Guide*.

When developing a canvas app, and you want to include functionality from Open CTI, do the following:

1. Include the Open CTI API in `index.jsp`.
2. Call `Sfdc.canvas.client.signedrequest()` to store the signed request needed by the console integration toolkit API. For example, if the Canvas method of authentication is a signed request, do the following:

   ```
   Sfdc.canvas.client.signedrequest('<%=signedRequest%>')
   ```

   If the Canvas method of authentication is OAuth, do the following in the callback function used to get the context, as shown in the *Canvas Developer Guide*:

   ```
   Sfdc.canvas.client.signedrequest(msg)
   ```

Consider the following when working with Open CTI and canvas apps:

- The Open CTI API script depends on the signed request and should be added after the call to `Sfdc.canvas.client.signedrequest()` has executed. We recommend that you load the scripts dynamically.
- To retrieve the entity ID of the record that is associated with the canvas sidebar component, do the following:

  ```
  // Get signedRequest
  var signedRequest = Sfdc.canvas.client.signedrequest();
  var parsedRequest = JSON.parse(signedRequest);
  // get the entity Id that is associated with this canvas sidebar component.
  var entityId = parsedRequest.context.environment.parameters.entityId;
  ```

- To retrieve the `entityId` for OAuth, do the following:

  ```
  var entityId = msg.payload.environment.parameters.entityId;
  ```

To see an example on how to retrieve `msg.payload`, see the *Canvas Developer Guide*.

SEE ALSO:

Salesforce Canvas Developer Guide: Getting Context in Your Canvas App

Salesforce Help: Add Console Components to Apps in Salesforce Classic

Work with the Console APIs for Open CTI

There are console-specific methods that you can use to interact with Open CTI. Use the Salesforce Console Integration Toolkit JavaScript APIs to interact with Salesforce Classic console apps.

For more information, see the Methods for Computer-Telephony Integration (CTI) in the *Salesforce Console Developer Guide*.

Best Practices

When working with Open CTI, keep the following best practices in mind.

- Since many of the methods in Open CTI are asynchronous and return their results using a callback method, Salesforce recommends that you refer to the documentation for each method to understand the information for each response.
- Errors generated by Open CTI are typically emitted in a way that doesn't halt JavaScript processing. We recommend that you use browser built-in developer tools to monitor the JavaScript console and to help you debug your code.
- If you plan on customizing, extending, or integrating the sidebars of the Salesforce console using Visualforce, review the online help for information about console components.

SEE ALSO:

Salesforce Help: Customize a Console with Custom Components in Salesforce Classic

CHAPTER 4 Methods for Lightning Experience

If your org is using Lightning Experience, use methods that work with Lightning Experience.

Important: Open CTI is in maintenance mode and is scheduled for retirement in February 2028. No new features or enhancements are being added to Open CTI. Effective immediately, Open CTI is deprecated and unavailable for newly created Agentforce Service orgs. To ensure long-term compatibility and access to the latest innovations, we recommend transitioning your development efforts to Service Cloud Voice. For more information, see this Knowledge Article.

To enable your contact center users to take advantage of the latest phone channel innovations, Salesforce recommends that you modernize your experience by moving to Service Cloud Voice. Service Cloud Voice offers many of the Open CTI features that you love and more. Unlike Open CTI, Service Cloud Voice is natively integrated with Omni-Channel and Command Center for Service, providing a seamless experience for contact center reps and supervisors across all digital channels. See Get to Know Service Cloud Voice.

Note: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience.You can't swap the two Open CTI APIs in custom JavaScript code because they behave and function differently. Make sure that you think about where you want to implement your CTI system before you begin developing.

SEE ALSO:

disableClickToDial() for Lightning Experience for Lightning Experience

Usage

Disables click-to-dial. This method is available in API version 38.0 or later.

Note: You can use this method with the Lightning web component `lightning-click-to-dial`. You can also use it with the Aura component `lightning:clickToDial`. Keep in mind that you can't use either component in iFrames. This method can't be used with the Visualforce component `support:clickToDial`.

Syntax

```
sforce.opencti.disableClickToDial({
     callback: function //Optional
})
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
```

```
        } else {
           console.error('Something went wrong! Errors:', response.errors);
        }
      };

      function disableClickToDial() {
          sforce.opencti.disableClickToDial({callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="disableClickToDial();">disableClickToDial()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

Lightning Components Developer Guide: `lightning:clickToDial`

enableClickToDial() for Lightning Experience for Lightning Experience

Usage

Enables click-to-dial. This method is available in API version 38.0 or later.

Note: You can use this method with the Lightning web component `lightning-click-to-dial`. You can also use it with the Aura component `lightning:clickToDial`. Keep in mind that you can't use either component in iFrames. This method can't be used with the Visualforce component `support:clickToDial`.

Syntax

```
sforce.opencti.enableClickToDial({
     callback: function //Optional
})
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function enableClickToDial() {
          sforce.opencti.enableClickToDial({callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="enableClickToDial();">enableClickToDial()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

Lightning Components Developer Guide: `lightning:clickToDial`

getAppViewInfo() for Lightning Experience for Lightning Experience

Usage

Returns information about the current application view. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.getAppViewInfo({
     callback: function
});
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function getAppViewInfo() {
          sforce.opencti.getAppViewInfo({callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="getAppViewInfo();">getAppViewInfo()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

When you switch from a record detail page to a list view, this method returns:

- In Lightning Experience, only the `url`
- In Lightning Experience console apps, nothing is returned because the listener isn't invoked

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.

`returnValue`	object	Returns the URL of the current application view and includes any applicable record ID, record name, and object type. For example:
		`{ "url": "https://MyDomainName.my.salesforce.com/lightning/r/Case/500D000000Ztcch1AA/view?t=1459806895555", "recordId": "001x0000003DGQR", "recordName": "Acme", "objectType": "Account" }`
		Invoking this API method on a person account object returns the following additional information.
		• `accountId` or `contactId`—The associated account or contact ID.
		• `personAccount`—Which is `true` if person accounts are enabled in your org, `false` otherwise.
		For example:
		`{ "url": "https://MyDomainName.my.salesforce.com/lightning/r/Account/001x0000003DGQR/view", "recordId": "001x0000003DGQR", "recordName": "Acme Person Account", "objectType": "Account", "contactId": "003D000000QOMqg", "personAccount": true }`
		Note: Since the URL structure of the `returnValue` can change in the future, we recommend that you don't build any logic based on it.

`error`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

getCallCenterSettings() for Lightning Experience for Lightning Experience

Usage

Returns the call center settings associated with the current user. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.getCallCenterSettings({
     callback: function
});
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function getCallCenterSettings() {
          sforce.opencti.getCallCenterSettings({callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="getCallCenterSettings();">getCallCenterSettings()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	If the API call is successful, the call center settings are returned.


```
{
  "/displayNameLabel":"Display Name",
  "/internalNameLabel":"InternalName",
  "/label":"Demo Call Center Adapter",
  "/reqDialingOptions/label":"Dialing Options",
  "/reqDialingOptions/reqInternationalPrefix":"01",

"/reqDialingOptions/reqInternationalPrefix/label":"International
 Prefix",

"/reqDialingOptions/reqInternationalPrefix/sortOrder":"2.0",

  "/reqDialingOptions/reqLongDistPrefix":"1",
  "/reqDialingOptions/reqLongDistPrefix/label":"Long
Distance Prefix",
  "/reqDialingOptions/reqLongDistPrefix/sortOrder":"1.0",

  "/reqDialingOptions/reqOutsidePrefix":"9",
  "/reqDialingOptions/reqOutsidePrefix/label":"Outside
 Prefix",
  "/reqDialingOptions/reqOutsidePrefix/sortOrder":"0.0",

  "/reqDialingOptions/sortOrder":"1.0",
  "/reqGeneralInfo/label":"General Information",
  "/reqGeneralInfo/reqAdapterUrl":"/apex/ContactCenter",

  "/reqGeneralInfo/reqAdapterUrl/label":"CTI Adapter
URL",
  "/reqGeneralInfo/reqAdapterUrl/sortOrder":"2.0",
  "/reqGeneralInfo/reqDisplayName":"Demo Call Center
Adapter",
  "/reqGeneralInfo/reqInternalName":"OpenCTI",

"/reqGeneralInfo/reqSalesforceCompatibilityMode":"Classic_and_Lightning",


"/reqGeneralInfo/reqSalesforceCompatibilityMode/label":"Salesforce
 Compatibility Mode",

"/reqGeneralInfo/reqSalesforceCompatibilityMode/sortOrder":"8.0",

  "/reqGeneralInfo/reqSoftphoneHeight":"550",
  "/reqGeneralInfo/reqSoftphoneHeight/label":"Softphone
 Height",
```

Name	Type	Description
		"/reqGeneralInfo/reqSoftphoneHeight/sortOrder":"6.0", "/reqGeneralInfo/reqSoftphoneWidth":"400", "/reqGeneralInfo/reqSoftphoneWidth/label":"Softphone Width", "/reqGeneralInfo/reqSoftphoneWidth/sortOrder":"7.0", "/reqGeneralInfo/reqStandbyUrl":"/apex/ContactCenter", "/reqGeneralInfo/reqStandbyUrl/label":"CTI Adapter URL2", "/reqGeneralInfo/reqStandbyUrl/sortOrder":"3.0", "/reqGeneralInfo/reqTimeout":"10000", "/reqGeneralInfo/reqTimeout/label":"Timeout", "/reqGeneralInfo/reqTimeout/sortOrder":"4.0", "/reqGeneralInfo/reqUseApi":"true", "/reqGeneralInfo/reqUseApi/label":"Use CTI API", "/reqGeneralInfo/reqUseApi/sortOrder":"5.0", "/reqGeneralInfo/sortOrder":"0.0", "/reqPhoneDemoSettings/label":"Phone Demo Settings", "/reqPhoneDemoSettings/reqIncomingNumber":"(415) 555-1212 (tel:4155551212)", "/reqPhoneDemoSettings/reqIncomingNumber/label":"Simulated Incoming Phone Number", "/reqPhoneDemoSettings/reqIncomingNumber/sortOrder":"0.0", "/reqPhoneDemoSettings/reqProvider":"DummyProvider", "/reqPhoneDemoSettings/reqProvider/label":"CTI Provider", "/reqPhoneDemoSettings/reqProvider/sortOrder":"1.0", "/reqPhoneDemoSettings/reqProviderAccount":"AXXXXXXXXXXXXXXXXX", "/reqPhoneDemoSettings/reqProviderAccount/label":"Provider Account", "/reqPhoneDemoSettings/reqProviderAccount/sortOrder":"2.0", "/reqPhoneDemoSettings/reqProviderAuthToken":"YYYYYYYYYYYYYYYYY", "/reqPhoneDemoSettings/reqProviderAuthToken/label":"Provider Auth Token", "/reqPhoneDemoSettings/reqProviderAuthToken/sortOrder":"3.0", "/reqPhoneDemoSettings/reqProviderCallerNumber":"415555555",

Name	Type	Description

		<pre>"/reqPhoneDemoSettings/reqProviderCallerNumber/label":"Provider Caller Number",
		"/reqPhoneDemoSettings/reqProviderCallerNumber/sortOrder":"4.0",
		"/reqPhoneDemoSettings/sortOrder":"2.0"
		}</pre> If the API call fails, null is returned.

`error`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

getSoftphoneLayout() for Lightning Experience for Lightning Experience

Usage

Returns the softphone layout of the current user. This method is available in API version 38.0 or later.

Note: The Open CTI for Lightning Experience methods `screenPop()` and `searchAndScreenPop()` don't support `screenPopSettings`.

Syntax

```
sforce.opencti.getSoftphoneLayout({
     callback: function
});
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            alert(response.returnValue);
```

```
        } else {
            console.error(response.errors);
            alert(
              'Something went wrong. Please check error information in developer console.'

            );
        }
    };

    function getSoftphoneLayout() {
        sforce.opencti.getSoftphoneLayout({
            callback: callback
        });
    }
  </script>
</head>
<body>
  <button onclick="getSoftphoneLayout();">Get Softphone Layout</button>
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	If the API call is successful, the value `true` is returned and the softphone layout definition is returned in `returnValue`, false otherwise.
`returnValue`	object	If the API call is successful, the softphone layout definition is returned. If the API call fails, `null` is returned. The returned object contains three elements that represent each of the call-types: • `"Internal"` • `"Inbound"` • `"Outbound"` Each call-type contains three sub-sections: • `"callRelatedFields"`—An array of call-related fields selected to display. Possible values are `"ANI"`, `"DNIS"`, `"SEGMENT"`, and `"QUEUE"`. • `"objects"`—The set of Salesforce objects selected to display, along with the `Field Label` and `Field Name` (API name) selected to display from each object. • `"screenPopSettings"`—This object contains a `"screenPopsOpenWithin"` field with a value of either `"ExistingWindow"` or `"NewWindow"`. Additionally, it contains the settings for each of the screen pop match types: `"NoMatch"`, `"SingleMatch"`, `"MultipleMatches"`. Each match

Name	Type	Description
		type contains a corresponding `"screenPopType"` field and may also contain a `"screenPopData"` field. – If `"screenPopType"` has a value of `"PopToEntity"`, then `"screenPopData"` contains the name of the target object. – If `"screenPopType"` has a value of `"PopToVisualforce"`, then `"screenPopData"` contains the name of the target Visualforce page. – If `"screenPopType"` has a value of `"PopToSearch"`, then there won't be a `"screenPopData"` field. – If `"screenPopType"` has a value of `"PopToFlow"`, then `"screenPopData"` contains the name of the target flow. The following is an example of a `returnValue`: `{ "Internal" : { "callRelatedFields" : ["ANI", "DNIS",] "objects" : { "User" : [{ "displayName" : "Name", "apiName" : "Name" }] }, "screenPopSettings" : {} }, "Inbound" : { "callRelatedFields" : ["ANI", "DNIS", "SEGMENT", "QUEUE"], "objects" : { "Account" : [{ "displayName" : "Account Name", "apiName" : "Name" }] }, "screenPopSettings" : { "NoMatch" : { "screenPopType" : "PopToEntity", "screenPopData" : "Contact" }, "SingleMatch" : {` … `}` `}`

Name	Type	Description

		<pre>"screenPopType" : "PopToVisualforce",
		"screenPopData" : "Visualforce_Page_Name"
		},
		"MultipleMatches" : {
		"screenPopType" : "PopToSearch"
		}
		}
		},
		"Outbound" : {
		"callRelatedFields" : [
		"DNIS"
		],
		"objects" : {
		"Account" : [{
		"displayName" : "Account Name",
		"apiName" : "Name"
		}
		]
		},
		"screenPopSettings" : {}
		}
		}</pre>

`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

Lightning Flow for Service Developer Guide (English only)

Sales Engagement Methods for Lightning Experience

Usage

These two methods allow your CTI implementation to communicate with Sales Engagement to handle Sales Engagement work. The `onWorkStart` method lets you listen for when a Sales Engagement work item starts. You communicate back to Sales Engagement with the `completeWork` method when the work completes. These methods are available in API version 46.0 or later.

Note: You can use this method with the Lightning web component `lightning-click-to-dial`. You can also use it with the Aura component `lightning:clickToDial`. Keep in mind that you can't use either component in iFrames. This method can't be used with the Visualforce component `support:clickToDial`.

Note: `CallEnd` is deprecated as of API version 48.0.

Syntax

The `onWorkStart` method lets you listen for when work starts. Sales Engagement calls your listener function when the work starts. In your listener, the `completeWorkWhen` parameter specifies whether to call Sales Engagement back when the task is saved (`TaskSaved`).

```
sforce.opencti.hvs.onWorkStart({listener: function});

{ // Listener function payload
  workId: string,            // Id of the current work
  completeWorkWhen: string,  // 'TaskSaved'
  attributes: {
    to: string               // Used to match to click_to_call
  }
}
```

When the work completes (when the task is saved), call the `completeWork` method so that the Sales Engagement system can update its information.

```
sforce.opencti.hvs.completeWork({
  workId: string,          // Id from onWorkStart
  attributes: {
    disposition: string,   // Only needed for task
    taskId: string,
    wasConnected: boolean, // Should be added, otherwise defaults to true
  },
  callback: function
});
```

Note: Only call this method for outbound calls. Inbound calls aren't started by Sales Engagement.

Arguments

Arguments for `onWorkStart`:

Name	Type	Description
`listener`	function	Function that is called when the work is started. This function's payload includes `workId`, `completeWorkWhen`, `attributes`, and `to`.
`workId`	string	ID of the work item. Depending on the type of work, this value is either the step tracker ID or the my list ID.
`completeWorkWhen`	string	Defines when to call the `completeWork` method. This value is `TaskSaved`.
`attributes`	object	Attributes object that contains the `to` field.
`to`	string	This field is used to link to the number in `onClickToDial`.

Arguments for `completeWork`:

Name	Type	Description
`workId`	string	ID of the work item. Use the same value that you previously received in `onWorkStart`.
`attributes`	object	Attributes object that can contain the following fields: `disposition`, `taskId`, `wasConnected`.
`disposition`	string	Value of the disposition. Required when `completeWorkWhen` is `TaskSaved`.
`taskId`	string	ID of the task. Required when `completeWorkWhen` is `TaskSaved`.
`wasConnected`	boolean	Indicates whether the call successfully connected. We recommend always passing this value. If not passed, the value defaults to true.
`callback`	function	Function executed when the call completes.

Sample Code

Listen for Sales Engagement work to start:

```
sforce.opencti.hvs.onWorkStart({
    listener: function(payload) {
        var workId = payload.workId;              // Save the work ID
        var whenVal = payload.completeWorkWhen;   // Save the completion requirement
        var toVal = payload.attributes.to;        // Save the number to associate with
onClickToDial
    }
});
```

Call Sales Engagement when a task is saved:

```
sforce.opencti.hvs.completeWork({
  workId: a07B0000006VFHrIAO,      // Id sent via onWorkStart
  attributes: {
      disposition: 'Completed',    // Disposition value
      taskId: '00TR00000032yfVMAQ' // ID of task created
      wasConnected: true           // Whether the call successfully connected
  },
  callback: function() { /* perform cleanup here */ }
});
```

SEE ALSO:

Salesforce Help: Sales Engagement

Salesforce Help: Phone Integration Considerations for Sales Engagement

isSoftphonePanelVisible() for Lightning Experience for Lightning Experience

Usage

Use this method to return the visibility status of the softphone panel. Returns `true` if the softphone panel is visible and `false` if it's minimized. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.isSoftphonePanelVisible({
     callback: function
});
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript with a callback

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function isSoftphonePanelVisible() {
          sforce.opencti.isSoftphonePanelVisible({callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="isSoftphonePanelVisible();">isSoftphonePanelVisible()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	Contains the boolean property `visible` which indicates the visibility status of the softphone panel. It's `true` if the softphone is visible and `false` if it's minimized.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

notifyInitializationComplete() for Lightning Experience for Lightning Experience

Usage

Notifies Salesforce that the softphone initialization is complete and that Salesforce should not switch to a standby URL. While the softphone initializes, a loading icon displays in the softphone area. To use a standby URL, you must specify the `reqStandbyUrl` and `reqTimeout` fields, in the call center's definition file. For more information, see Optional Call Center Elements and Attributes

The `notifyInitializationComplete()` method for Lightning Experience works differently from the Salesforce Classic method.

- In Lightning Experience, the method takes an optional callback object. In Salesforce Classic, the method doesn't take arguments.
- In Lightning Experience, after you go to the standby URL your browser session continues to use that standby URL. To force the standby URL check, you must restart the browser—close it and open it again. In Salesforce Classic, the standby URL check was completed only after you logged in to Salesforce, and the check wasn't repeated if you kept using the same Salesforce session.

Syntax

```
sforce.opencti.notifyInitializationComplete({
     callback: function //Optional
});
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="http://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      // Invokes API method
      sforce.opencti.notifyInitializationComplete();
    </script>
  </head>
  <body>
    The Open CTI framework was notified that the softphone initialization is complete.
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

onClickToDial() for Lightning Experience for Lightning Experience

Usage

Registers a function to call when a user clicks an enabled phone number. This method is available in API version 38.0 or later.

Note: You can use this method with the Lightning web component `lightning-click-to-dial`. You can also use it with the Aura component `lightning:clickToDial`. Keep in mind that you can't use either component in iFrames. This method can't be used with the Visualforce component `support:clickToDial`.

Syntax

```
sforce.opencti.onClickToDial({
     listener: function
})
```

Arguments

Name	Type	Description
`listener`	function	JavaScript method called when the user clicks an enabled phone number.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api//lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var listener = function(payload) {
        console.log('Clicked phone number: ' + payload.number);
      };

      // Register the listener.
      window.addEventListener('load', function() {
        sforce.opencti.onClickToDial({listener: listener});
      });
   </script>
  </head>
</html>
```

Payload

The payload object passed to each call to the listener method contains the following fields.

Name	Type	Description
`number`	number	Provides the phone number clicked by the user.
`recordId`	string	Provides the ID of the record associated with the clicked phone number.
`recordName`	string	Provides the name of the record for the clicked phone number.
`objectType`	string	Provides the type of record associated with the clicked phone number.
`accountId` or `contactId`	string	If the clicked phone number belongs to a person account, the associated account or contact ID is provided.
`personAccount`	boolean	If the clicked phone number belongs to a person account, this property is `true`. If person accounts aren't enabled in your org, this field isn't included in the payload object.

Name	Type	Description
`params`	string	Comma-separated list of parameters associated with the phone number passed into the component.

SEE ALSO:

Lightning Components Developer Guide: `lightning:clickToDial`

onNavigationChange() for Lightning Experience for Lightning Experience

Usage

Registers a function to call when one of the following actions occur:

- The URL of the page changes
- In a standard app in Lightning Experience, a user navigates between browser tabs or windows and the document comes back into focus. If the document doesn't come back into focus, the listener isn't invoked. Also, internal navigation actions that open a new record modal, such as the screenPop() method, will invoke the listener.
- In a console app in Lightning Experience, a user navigates between primary tabs and subtabs in a console, or the document comes back into focus when a user navigates between browser tabs or windows.
- In a console app, a tab is refreshed, such as when you create a new case in a console and save it.

For example, the listener is invoked when a user navigates away from a browser tab and then comes back to the tab.

This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.onNavigationChange({
     listener: function
});
```

Arguments

Name	Type	Description
`listener`	function	JavaScript method called upon a navigation change.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var listener = function(payload) {
```

```
        console.log('Navigation change occurred. Payload: ', payload);
      };

      // Register the listener.
      window.addEventListener('load', function() {
        sforce.opencti.onNavigationChange({listener: listener});
      });
    </script>
  </head>
</html>
```

Payload

The payload object passed to each call to the listener method contains the following fields.

When you switch from a record detail page to a list view, this method returns:

- In Lightning Experience, only the `url`
- In Lightning Experience console apps, nothing is returned because the listener isn't invoked

Name	Type	Description
`url`	string	Provides the URL of the page the user navigated to.
`recordId`	string	If the user navigated to a Salesforce record, such as an account or case, the loaded record ID is returned. Otherwise, the field is empty.
`recordName`	string	If the user navigated to a Salesforce record, the loaded record name. Otherwise, the field is empty.
`objectType`	string	If the user navigated to a Salesforce record, the loaded object type, such as an account or case. Otherwise, the field is empty.
`accountId` or `contactId`	string	If the page the user navigated to is the record home of a person account, the associated account or contact ID is returned.
`personAccount`	boolean	Returned only if person accounts are enabled in your org and the user navigates to a person account. If the page the user navigated to is the record home of a person account, this field is `true`. If the page the user navigated to is not the record home of a person account, this field is `false`. If the page the user navigated to is a business account, this field along with the `accountId` and `contactId` fields aren't included in the payload.

refreshView() for Lightning Experience for Lightning Experience

Usage

Returns `true` if view refresh is invoked, `false` otherwise. When this method is called within the Salesforce console, it refreshes the current active view. If any related lists are included in this tab, they're refreshed too. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.refreshView({
     callback:function
});
```

Arguments

Name	Type	Description
`callback`	function	Optional. JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript without a callback

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
 var param = {};
        function refreshView() {
                sforce.opencti.refreshView(param);
         }
</script>
</head>
<body>
       <button onclick="refreshView();">refreshView</button>
</body>
</html>
```

Sample Code–HTML and JavaScript with a callback

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
 var param = {};
             var callback = function(response) {
         if (response.success) {
```

```
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
       };
            param.callback = callback;

       function refreshView() {
                sforce.opencti.refreshView(param);
        }
</script>
</head>
<body>
       <button onclick="refreshView();">refreshView</button>
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

runApex() for Lightning Experience for Lightning Experience

Usage

Executes an Apex method from an Apex class that's exposed in Salesforce. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.runApex({
    apexClass:string,
    methodName:string,
    methodParams:string,
    callback:function //Optional
})
```

Arguments

Name	Type	Description
`args`	object	A JavaScript object containing the following: • `apexClass` (string)—Specifies the Apex class of the method to execute. • `methodName` (string)—Specifies the method to execute. • `methodParams` (string)—Specifies the method parameters to pass. The string must include field value pairs and be formatted as a valid query string. For example: `name=acme&phone=(212) 555-5555` If the Apex method doesn't take any parameters, you can specify `methodParams` as `none` or an empty object, `{}`. • `callback` (function)—JavaScript method called upon completion of the method.

Sample Code–HTML and JavaScript

1. In Setup, create an Apex class and Apex method.

```
global class AccountRetrieval{

webService static String getAccount(String name) {
   List<Account> accounts = new List<Account>();
   for (Account account : Database.query('Select Id, Name, phone from Account where Name
 like \'' + name + '%\'')){
       accounts.add(account);
   }
       String JSONString = JSON.serialize(accounts);
       return JSONString;
   }
}
```

2. In Setup, click **Generate from WSDL** to expose the method and class so that a third-party softphone can call it.
3. Add your code to the softphone:

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
       var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
```

```
      };
       function runApex() {
          var param = {apexClass: 'AccountRetrieval', methodName: 'getAccount',
methodParams: 'name=acme'};
          param.callback = callback;
          //Invokes API method
          sforce.opencti.runApex(param);
       }
   </script>
</head>
<body>
       <button onclick="runApex();">runApex</button>
</body>
</html>
```

4. Output is returned. In this example, one account named Acme was found:

```
{
    "success": true,
    "returnValue": {
        "runApex":
"[{\"attributes\":{\"type\":\"Account\",\"url\":\"/services/data/v38.0/sobjects/Account
/001xx000003DGawAAG\"},\"Id\":\"001xx000003DGawAAG\",\"Name\":\"Acme\",\"Phone\":\"(212)
 555-5555\"}]"
    },
    "errors": null
}
```

Response

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	A JavaScript object containing the result from executing the method from the specified Apex class. No specific format is returned. The format is determined by the value from the method that executed. For example: `{"runApex":"[{\"attributes\":{\"type\":\"Account\", \"url\":\"/services/data/v66.0/sobjects/Account/ 001xx000003DGawAAG\"},\"Id\":\"001xx000003DGawAAG\", \"Name\":\"Acme\",\"Phone\":\"(212)555-5555\"}]"}`
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

Salesforce Help: Apex Code Overview

saveLog() for Lightning Experience for Lightning Experience

Usage

Saves or updates an object in Salesforce. This method is available in API version 38.0 or later.

Note:

- To update using this method, include `Id`.
- To create using this method, include `entityApiName`.
- If an object uses `recordType`, pass the `recordTypeId` in the `saveLog` call. If you don't pass the `recordType`, the record is created using the default `recordType` for the profile. To create a person account, you can pass the person account `recordType` if the profile's default is to a business account.
- To refresh after you update or create using this method, call the `refreshView` method in the `callback` method.

Syntax

```
sforce.opencti.saveLog({
    value:{
        entityApiName:string, //Optional
        Id:string, //Optional
        param:value //Optional
        },
    callback:function //Optional
})
```

Arguments

Name	Type	Description

`value`	object	Specifies the fields to save or update on the object.
		If the object's ID is specified, a record is updated. For example:
		`{Id:"00QR0000000yN5iMAE", LastName:"New lastname" }`
		If the object's ID isn't specified, a new record is created. For example:
		`{entityApiName:"Contact", LastName:"LastName" },callback:callback}`
		Note: To create a record, ensure all the required fields are specified.

`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
      var callback = function (response) {
         if (response.result) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      }
      function saveLog() {
         //Update an existing object with the ID specified
         sforce.opencti.saveLog({value:{Id:"00QR0000000yN5iMAE", LastName:"New lastname"
}, callback:callback});
         //Create a contact
         sforce.opencti.saveLog({value:{entityApiName:"Contact", LastName:"LastName"
},callback:callback});
         //Update a lead
         sforce.opencti.saveLog({value:{Id:"00QR0000000yN5iMAE", LastName:"New lastname"
},callback:callback});
      }
   </script>
</head>
<body>
   <button onclick="saveLog();">saveLog</button>
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	ID of object if creating or updating the object was successful; `null` if creating or updating the object wasn't successful.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

screenPop() for Lightning Experience for Lightning Experience

Usage

Pops to a new location as specified by the input type and parameters. This method is available in API version 38.0 or later.

Note: Open CTI for Lightning Experience doesn't support the softphone layout field `Screen pops open within` when the value is `New browser window or tab`. In Lightning Experience, the default value is `Existing browser window`.

Syntax

```
sforce.opencti.screenPop({
     type: sforce.opencti.SCREENPOP_TYPE.*, //Review the arguments section.
     params: object //Depends on the SCREENPOP_TYPE. Review the arguments section.
 });
```

Arguments

Name	Type	Description
`type`	string	The enumerated type to screen pop to. Use one of the following values: • `sforce.opencti.SCREENPOP_TYPE.SOBJECT` • `sforce.opencti.SCREENPOP_TYPE.URL` • `sforce.opencti.SCREENPOP_TYPE.OBJECTHOME` • `sforce.opencti.SCREENPOP_TYPE.LIST` • `sforce.opencti.SCREENPOP_TYPE.SEARCH` • `sforce.opencti.SCREENPOP_TYPE.NEW_RECORD_MODAL` • `sforce.opencti.SCREENPOP_TYPE.FLOW`

`params`	object	An object holding arguments depending on the type.
		• For `SOBJECT`:
		`params : { recordId: string }`
		Where `recordId`, is the ID of the standard or custom object in Salesforce.
		• For `URL`:
		`params : { url: string }`
		The URL must be a relative parameter. For more information about the `URL`, see the `force:navigateToURL` `url` parameter in the *Lightning Aura Components Developer Guide*.
		• For `OBJECTHOME`:
		`params : { scope: string }`

Name	Type	Description

		Pops to the home of an object or entity such as a case or account. For more information about the `scope`, see the `force:navigateToSObject recordID` parameter in the *Lightning Aura Components Developer Guide*. Here's a sample input:
		`{ scope: "Account"}`
		• For `LIST`:
		`{ listViewId: string, scope: string }`
		For more information about the `listViewId` and `scope` parameter, see `force:navigateToList` in the *Lightning Aura Components Developer Guide*.
		• For `SEARCH`:
		`params : {searchString: string}`
		Pops to the Top Results section of the search page. The string must be at least 3 characters in length.
		• For `NEW_RECORD_MODAL`:
		`params : {entityName: string, //required defaultFieldValues: object//optional }`
		Required. The API name of the custom or standard object, such as Account, Case, Contact, or Lead.
		Note: For custom objects, the name for a new record model follows this format:
		`objectName__c`
		This name takes the default namespace. Notice that the separator includes 2 underscores.
		If your org has namespace enabled, you must prefix it for custom objects. Use this format:
		`namespace__objectName__c`
		To pop to a new person account model, use the input `Account`.
		• For `FLOW`:
		`params: {flowDevName: 'Flow_Dev_Name', flowArgs: [{'name': 'Name', 'type': 'Type', 'value': 'Value'}]}`
		Pops to the target flow. Additional arguments can be passed to the flow, for example, the caller's phone number or a list of matching records.
		Note: Only active UI screen flows (that use the type `Flow`) can be popped. Arguments passed to the flow must be defined as variables inside the flow, or else they're ignored at runtime. Consult the FlowVariable

Name	Type	Description
		section in the topic on Flow in the Metadata API Developer Guide for details on flow arguments.
`defaultFieldValues`	object	Optional. If you set up your softphone to pop to a new entity when there are no search results (for inbound calls), you can use this argument to specify the default fields for the screen pop. For example, when the screen pop opens for the new entity it's pre-populated with the fields you've specified. `{ searchParams: 'searchTermWithEmptyResults', callType: 'inbound', defaultFieldValues: {LastName : 'searchAndScreenPopLastName'}, deferred: false, callback: function(result) { if (result.success) { console.log(result.returnValue); } else { console.log(result.errors); } }}`
`callback`	function	Optional. JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript with a callback

```
<html>
 <head>
   <script type="text/javascript"
src="http://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
  var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
       };
        function screenPop() {
                sforce.opencti.screenPop({type: sforce.opencti.SCREENPOP_TYPE.OBJECTHOME,
 params: {scope:"Account"}, callback: callback  });
         }
</script>
</head>
<body>
       <button onclick="screenPop();">screenPop</button>
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

Lightning Flow for Service Developer Guide (English only)

searchAndScreenPop() for Lightning Experience for Lightning Experience

Usage

Searches objects specified in the softphone layout for a given string. Returns search results and screen pops any matching records. This method respects screen pop settings defined in the softphone layout. This method is available in API version 38.0 or later.

Note: The returned response displays only matches that meet your softphone layout settings. However, the search page that screen pops, displays all matches, regardless of the objects you specify in your sofpthone layout.

The `searchAndScreenPop()` method for Lightning Experience works differently from the Salesforce Classic method.

- Open CTI for Lightning Experience doesn't support the softphone layout field `Screen pops open within` when the value is `New browser window or tab`. In Lightning Experience, the default value is `Existing browser window`.
- Open CTI for Lightning Experience provides a new argument called `deferred`.

Tip: The `searchAndGetScreenPopUrl()` method is not available in the Open CTI API for Lightning Experience. To accomplish the same functionality in Lightning Experience, use the `deferred` parameter available in this method. Pass the value in `SCREEN_POP_DATA` from the return object into the `screenPop ()` method.

If you're noticing inconsistent behavior with the default settings of your softphone layout, edit your softphone layout to force the cache to refresh. From Setup, edit your softphone layout and save the changes. Then edit the layout again and reset the layout to the default settings.

Syntax

```
sforce.opencti.searchAndScreenPop({
     searchParams:string //Optional
     queryParams:string, //Optional
     defaultFieldValues:object, //Optional
     callType:sforce.opencti.CALL_TYPE.*, //Required. See arguments for more information.

     deferred:boolean //Optional)
```

```
    callback:function //Optional
});
```

Arguments

Name	Type	Description
`searchParams`	string	String to search.
`queryParams`	string	Specifies the query parameters to pass to the URL. Query parameters are only passed to the URL if the screen pop option is set to `Pop to Visualforce`.
`params`	object	Specifies arguments to pass to a flow. `flowArgs` is a list of objects with key-value pairs containing information about the arguments being passed to the flow. Arguments must correspond to input variables of the specified name and type defined in the flow, else they are ignored. `params: { FLOW: { flowArgs: [{ 'name': 'Your_Name', 'type': 'Your_Type', 'value': 'Your_Value'}] } }` **Important:** This argument can only be used with flows. Consult the FlowVariable section in the topic on Flow in the Metadata API Developer Guide for details on flow arguments.
`defaultFieldsValue`	object	Optional. If you set up your softphone to pop to a new entity when there are no search results (for inbound calls), you can use this argument to specify the default fields for the screen pop. For example, when the screen pop opens for the new entity it's pre-populated with the fields you've specified. `{ searchParams: 'searchTermWithEmptyResults', callType: 'inbound', defaultFieldValues: {LastName : 'searchAndScreenPopLastName'}, deferred: false, callback: function(result) { if (result.success) { console.log(result.returnValue); } else { console.log(result.errors); } }`
`callType`	string	Specifies the type of call, such as inbound, outbound, internal, or null. Per the settings in the softphone layout, the call type determines which objects to search for any matches. Specify the call type with one of the following values: • `sforce.opencti.CALL_TYPE.INBOUND` • `sforce.opencti.CALL_TYPE.OUTBOUND` • `sforce.opencti.CALL_TYPE.INTERNAL`

Name	Type	Description
`deferred`	boolean	Specifies whether the screen pop is performed immediately following the search or if it's performed later. If the screen pop is performed later, a JSON object is returned. This object must be passed unmodified to `sforce.opencti.screenPop` to perform the operation. • `False`—Default value. Indicates an immediate screen pop after the search is performed. • `True`—A JSON {object} is returned in the `SCREEN_POP_DATA` key. Return this object unmodified to `sforce.opencti.screenPop` for a screen pop.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
   <script type="text/javascript">
        var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
       };
       function searchAndScreenPop() {
                //Invokes API method
                sforce.opencti.searchAndScreenPop({ searchParams : 'Acme',queryParams :
'Key1=value1&Key2=value2', callType : sforce.opencti.CALL_TYPE.INBOUND,  deferred: false,
 callback : callback });
        }
    </script>
</head>
<body>
    <button onclick="searchAndScreenPop();">searchAndScreenPop</button>
</body>
</html>
```

Response

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	Returns a list of objects that match the search results. The search is performed on the objects specified in the softphone layout. For each object found, the object ID,

Name	Type	Description
		object type, last modified date, field names, and field values are returned as JSON objects. **Note:** Make sure that you specify fields to return in your softphone layout. If you don't, nothing is returned. The following is an example of searching for "Acme," and finding one Account and three Opportunity objects: `{ "006x0000001ZcyG":{ "Id": "006x0000001ZcyG" "LastModifiedDate": "2017-10-24T18:42:39.000Z", "Name":"Acme - 600 Widgets", "RecordType":"Opportunity" }, "001x0000003DGQR":{ "Name":"Acme", "Type":"Analyst", "object":"Account", "displayName":"Company" }, "006x0000001ZcyH":{ "Id": "006x0000001ZcyH" "LastModifiedDate": "2017-10-24T18:42:39.000Z", "Name":"Acme - 200 Widgets", "RecordType":"Opportunity" }, "006x0000001ZcyF":{ "Id": "006x0000001ZcyF" "LastModifiedDate": "2017-10-24T18:42:39.000Z", "Name":"Acme - 1,200 Widgets", "RecordType":"Opportunity" } }`  Invoking this API method with a `deferred` parameter returns the following sample output without any screen navigation.  `{ "006x0000001ZcyG":{ "Id": "006x0000001ZcyG" "LastModifiedDate": "2017-10-24T18:42:39.000Z", "Name":"Acme - 600 Widgets", "RecordType":"Opportunity" }, SCREEN_POP_DATA : {} //an object. Do not modify it. Pass it to screenPop() API to navigate. }` Invoking this API method on an account or contact object returns additional information.

Name	Type	Description
		• `AccountId` or `PersonContactId`—The associated account or contact ID. • `IsPersonAccount`—`true` if the account or contact is a person account, `false` otherwise. For example: `{ "006x0000001ZcyG":{ "Id": "006x0000001ZcyG" "IsPersonAccount": true, "LastModifiedDate": "2017-11-15T00:10:47.000Z", "Name":"Acme Person Account", "PersonContactId" : "003D000000QOMqg", "RecordType":"Account" } }`
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

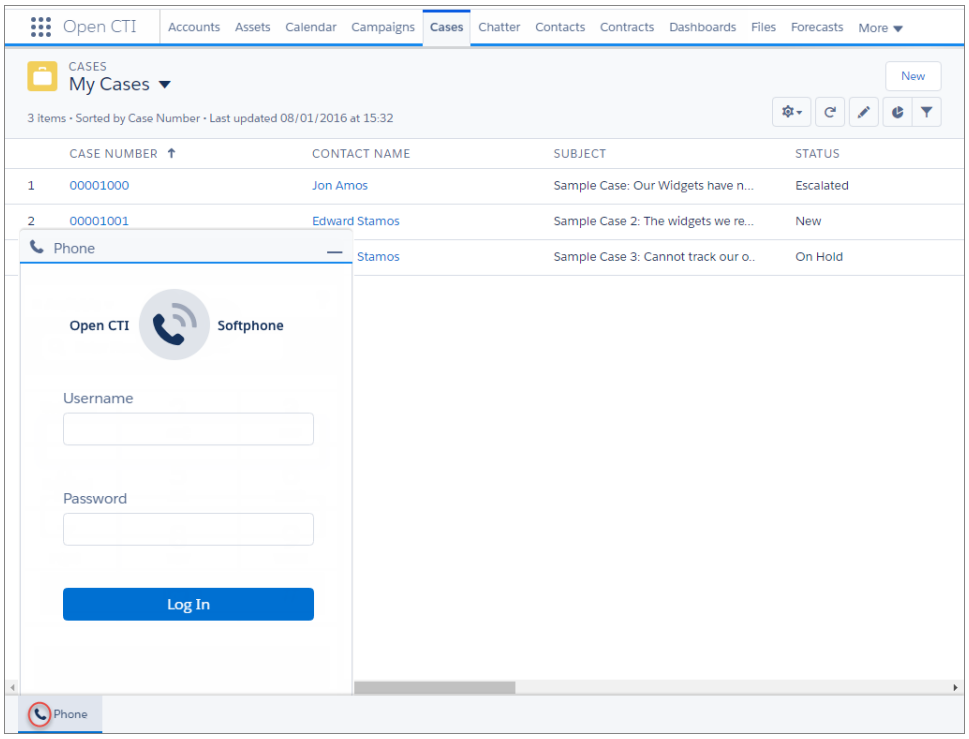
Lightning Flow for Service Developer Guide (English only)

setSoftphoneItemIcon() for Lightning Experience for Lightning Experience

Usage

Sets the icon for the softphone item in the utility bar. Returns `true` if the function is successfully executed, and `false` when there is a failure. This method is available in API version 38.0 or later.

The softphone icon in the utility bar.



Syntax

```
sforce.opencti.setSoftphoneItemIcon({
     key:key,
     callback:function  //Optional
});
```

Arguments

Name	Type	Description
`key`	string	The key corresponding to the icon in the Lightning Design System you want to use for the softphone icon in the utility bar.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
<head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
```

```
        if (response.success) {
           console.log('API method call executed successfully! returnValue:',
response.returnValue);
        } else {
           console.error('Something went wrong! Errors:', response.errors);
        }
      };

      function setSoftphoneItemIcon() {
          sforce.opencti.setSoftphoneItemIcon({key:"call", callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="setSoftphoneItemIcon();">setSoftphoneItemIcon()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

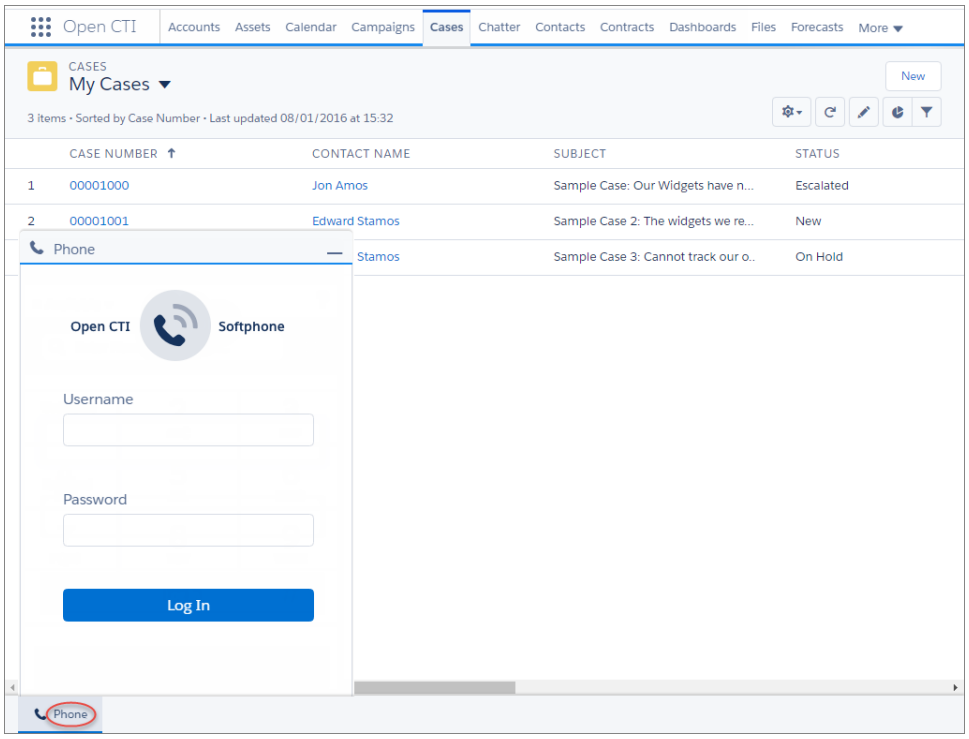
Salesforce Lightning Design System: Utility Icons

setSoftphoneItemLabel() for Lightning Experience for Lightning Experience

Usage

Sets the label for the softphone component item in the utility bar. Returns `true` if the function is successfully executed, and `false` when there is a failure. This method is available in API version 38.0 or later.

The softphone label in the utility bar.



Syntax

```
sforce.opencti.setSoftphoneItemLabel({
     label: string,
     callback:function //Optional
});
```

Arguments

Name	Type	Description
`label`	string	The string you want to use for the softphone label in the utility bar.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
```

```
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function setSoftphoneItemLabel() {
        sforce.opencti.setSoftphoneItemLabel({label: "MySoftphone", callback: callback});

      }
     </script>
  </head>
  <body>
    <button onclick="setSoftphoneItemLabel();">setSoftphoneItemLabel()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

setSoftphonePanelHeight() for Lightning Experience for Lightning Experience

Usage

Sets the softphone panel height in the utility bar. The height must be specified in pixels. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.setSoftphonePanelHeight({
     heightPX:height,
     callback:function  //Optional
});
```

Arguments

Name	Type	Description
`heightPX`	number	The softphone panel height in pixels. The height must be a number from 240 through 2,560.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
 <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function setSoftphonePanelHeight() {
          sforce.opencti.setSoftphonePanelHeight({heightPX: 400, callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="setSoftphonePanelHeight();">setSoftphonePanelHeight()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

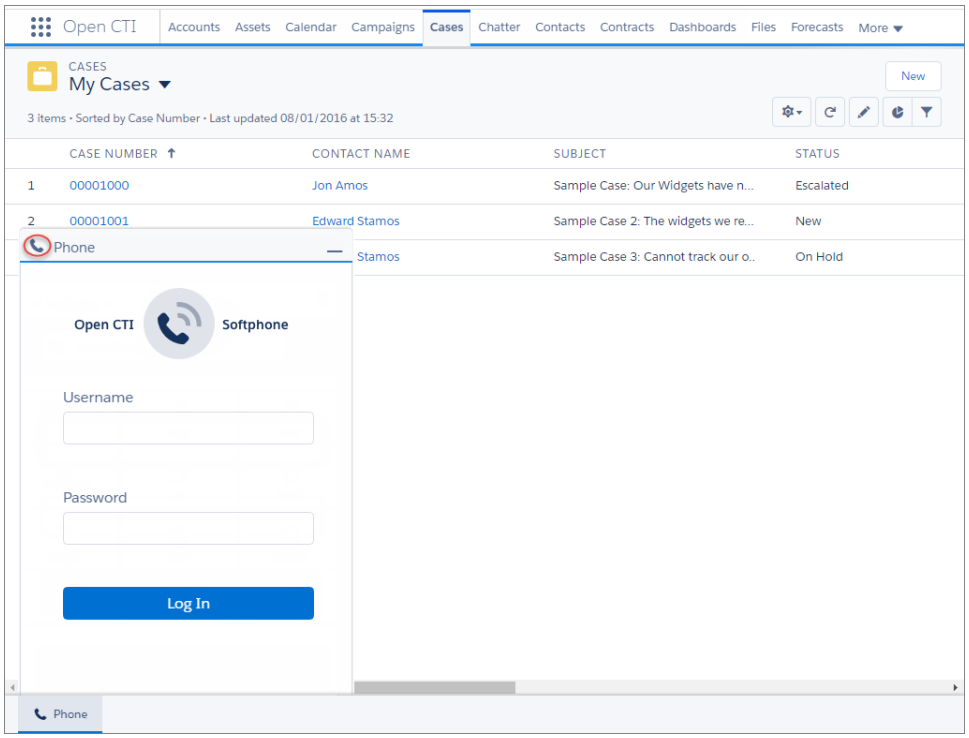
Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

setSoftphonePanelIcon() for Lightning Experience for Lightning Experience

Usage

Sets the icon for the softphone panel. Returns `true` if the function is successfully executed, and `false` when there is a failure. This method is available in API version 38.0 or later.

The softphone panel icon.



Syntax

```
sforce.opencti.setSoftphonePanelIcon({
     key:key,
     callback:function  //Optional
});
```

Arguments

Name	Type	Description
`key`	string	The key corresponding to the icon in the Lightning Design System you want to use for the softphone panel icon.

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function setSoftphonePanelIcon() {
          sforce.opencti.setSoftphonePanelIcon({key:"call", callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="setSoftphonePanelIcon();">setSoftphonePanelIcon()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

SEE ALSO:

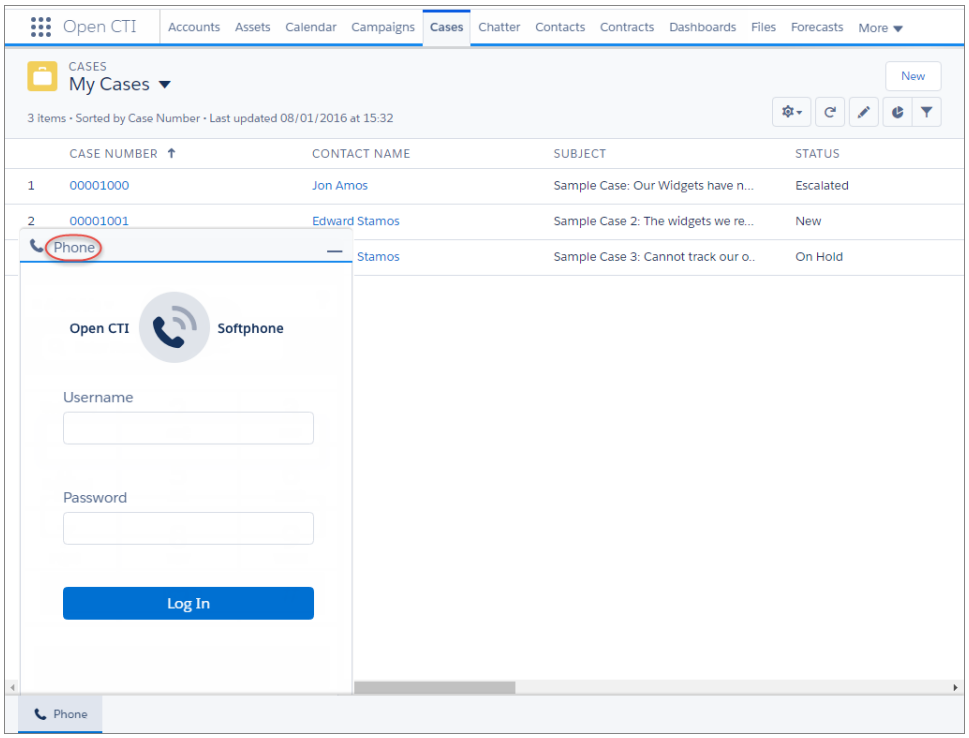
Salesforce Lightning Design System: Utility Icons

setSoftphonePanelLabel() for Lightning Experience for Lightning Experience

Usage

Sets the label for the softphone panel. Returns `true` if the function is successfully executed, and `false` when there is a failure. This method is available in API version 38.0 or later.

The softphone panel label.



Syntax

```
sforce.opencti.setSoftphonePanelLabel({
     label: string,
     callback:function //Optional
});
```

Arguments

Name	Type	Description
`label`	string	The string you want to use for the softphone panel label.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function setSoftphonePanelLabel() {
         sforce.opencti.setSoftphonePanelLabel({label: "Mysoftphone",callback: callback});

      }
     </script>
  </head>
  <body>
    <button onclick="setSoftphonePanelLabel();">setSoftphonePanelLabel()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

setSoftphonePanelVisibility() for Lightning Experience for Lightning Experience

Usage

Sets the visibility status of the softphone panel. When the `visible` parameter is passed as `true`, the softphone panel is displayed. When it's set to `false`, the panel is minimized. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.setSoftphonePanelVisibility({
   visible:true,
   callback:function  //Optional
});
```

Arguments

Name	Type	Description
`visible`	boolean	To dock (display) the softphone panel, set the value to `true`. To minimize (hide) the softphone panel, set the value to `false`.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
         }
      };

      function setSoftphonePanelVisibility() {
         sforce.opencti.setSoftphonePanelVisibility({visible: true, callback: callback});

      }
     </script>
  </head>
  <body>
   <button onclick="setSoftphonePanelVisibility();">setSoftphonePanelVisibility()</button>

  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

setSoftphonePanelWidth() for Lightning Experience for Lightning Experience

Usage

Sets the softphone panel width in the utility bar. The width must be specified in pixels. This method is available in API version 38.0 or later.

Syntax

```
sforce.opencti.setSoftphonePanelWidth({
   widthPX:width,
   callback:function  //Optional
});
```

Arguments

Name	Type	Description
`widthPX`	number	The softphone panel width in pixels. The height must be a number from 200 through 1,920.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–HTML and JavaScript

```
<html>
  <head>
    <script type="text/javascript"
src="https://domain:port/support/api/66.0/lightning/opencti_min.js"></script>
    <script type="text/javascript">
      var callback = function(response) {
         if (response.success) {
            console.log('API method call executed successfully! returnValue:',
response.returnValue);
         } else {
            console.error('Something went wrong! Errors:', response.errors);
```

```
        }
      };

      function setSoftphonePanelWidth() {
          sforce.opencti.setSoftphonePanelWidth({widthPX: 400, callback: callback});
      }
     </script>
  </head>
  <body>
    <button onclick="setSoftphonePanelWidth();">setSoftphonePanelWidth()</button>
  </body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`success`	boolean	Returns `true` if the API method call was invoked successfully, `false` otherwise.
`returnValue`	object	This API method doesn't return this object. The `returnValue` is always `null`.
`errors`	array	If the API call was successful, this variable is `null`. If the API call failed, this variable returns an array of error messages.

Lightning Message Service Methods for Lightning Experience

Usage

These three methods allow your CTI implementation to communicate with other Visualforce, Aura, and Lightning components that also use the Lightning Message Service. The `subscribe` method attaches a listener function to a specified Lightning Message Channel. The `publish` method lets you send a message on a Lightning Message Channel. The `unsubscribe` method lets you remove listener functions you added with the subscribe method.

These methods are available in API version 47.0 or later.

Note: These methods can be used only with Lightning Experience. They can be used in iFrame and Visualforce components.

Syntax

The `subscribe` method allows you to attach a listener function to a specified Lightning Message Channel. Lightning Message Service calls your listener function with the message that a component sends.

```
var SAMPLEMC = "SAMPLEMC__c";
var SAMPLEMC_SUBSCRIPTION = null;
sforce.opencti.subscribe({channelName: SAMPLEMC, listener: onPublishMessage, callback:
subscribeSampleMCCallback});
```

Arguments for `subscribe`

Name	Type	Description
`channelName`	string	The name of the Message Channel that you can subscribe to.
`listener`	function	JavaScript method that's called when a message is sent on the Message Channel.
`callback`	function	JavaScript method that's executed when the API method call is completed.

Arguments for `unsubscribe`

Name	Type	Description
`subscription`	object	The subscription from which you can remove the listener function.
`callback`	function	JavaScript method that's executed when the API method call is completed.

Arguments for `publish`

Name	Type	Description
`channelName`	string	The name of the Message Channel that you can subscribe to.
`message`	object	Serializable JSON object that is sent on the Message Channel.
`callback`	function	JavaScript method that's executed when the API method call is completed.

SEE ALSO:

- Blog: Lightning Message Service
- Sample Code for Using Lightning Message Service with Open CTI
- *Lightning Aura Components Developer Guide: Communicating Across the DOM with Lightning Message Service*
- *Visualforce Developer Guide: Communicating Across the DOM with Lightning Message Service*

Common Error Messages for Lightning Experience Methods

An error object is returned as an array for all Lightning Experience methods.

The following fields are contained as part of the error object.

`code: string`

A constant string denoting an error code.

`description: string`

A description of the error code.

`details: object`

Typically undefined. This constant can contain details about the error object for the `saveLog` method.

Sample error object:

```
[{
   code: code1
   description: description1
   details: details1
   },{
   code: code2
   description: description2
   details: details2
}]
```

Sample error object for the INVALID_PARAM error code:

```
[{
   code: "INVALID_PARAM",
   description: "An invalid value was passed to the parameter parameterName. A numeric
value was expected, but undefined was found instead."
}]
```

Sample error object for the GENERIC_PARAM error code:

```
[{
   code: "GENERIC_ERROR",
   description: "An error occurred while calling the API method."
}]
```

Sample error object for the SERVER_ERROR code:

```
[{
   code: "SERVER_ERROR",
   description: "A problem was encountered on the server."
}]
```

Sample error object for the SOFTPHONE_CONTAINER_ERROR code:

```
[{
   code: "SOFTPHONE_CONTAINER_ERROR",
   description: "Unable to execute sendPostMessage because the softphone container hasn't
 initialized yet."
}]
```

For the `runApex` method, if there is a server error, the `description` field provides `"Could not load Apex class: apexClassName."`

For the `saveLog` method, the `details` field provides information based on the type of error. For example:

```
[{
   code:"GENERIC_ERROR",
  description:"An error occurred while calling the saveLog() API method. Review the Details
 field in the error section.",
   details:[{
         message:"An error occurred while trying to update the record. Please try again.",
```

```
         pageErrors:[],
         fieldErrors:{
            Name:[{
                  statusCode:"REQUIRED_FIELD_MISSING",
                  message:"Required fields are missing: [Name]",
                  fieldLabel:"Account Name",
                  columnApiName:"Name"
               }]
            },
         potentialDuplicates:[]
      }]
}]
```

CHAPTER 5 Methods for Salesforce Classic

If your org is using Salesforce Classic, use methods that work with Salesforce Classic.

Important: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience.You can't swap the two Open CTI APIs in custom JavaScript code because they behave and function differently. Make sure that you think about where you want to implement your CTI system before you begin developing.

Methods for Salesforce Application Interaction

Open CTI lets your CTI system interact with the Salesforce application, including elements on a Case Feed page.

Methods for Computer-Telephony Integration (CTI)

Open CTI lets you integrate your CTI system with Salesforce.

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Method Parity Between Open CTI for Salesforce Classic and Lightning Experience

Methods for Salesforce Application Interaction

Open CTI lets your CTI system interact with the Salesforce application, including elements on a Case Feed page.

Important: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience.You can't swap the two Open CTI APIs in custom JavaScript code because they behave and function differently. Make sure that you think about where you want to implement your CTI system before you begin developing.

getPageInfo() for Salesforce Classic for Salesforce Classic

isInConsole() for Salesforce Classic for Salesforce Classic

isVisible() for Salesforce Classic for Salesforce Classic

notifyInitializationComplete() for Salesforce Classic for Salesforce Classic

onFocus() for Salesforce Classic for Salesforce Classic

onObjectUpdate() for Salesforce Classic for Salesforce Classic

refreshObject() for Salesforce Classic for Salesforce Classic

refreshPage() for Salesforce Classic for Salesforce Classic

refreshRelatedList() for Salesforce Classic for Salesforce Classic

reloadFrame() for Salesforce Classic for Salesforce Classic

runApex() for Salesforce Classic for Salesforce Classic

saveLog() for Salesforce Classic for Salesforce Classic
screenPop() for Salesforce Classic for Salesforce Classic
searchAndGetScreenPopUrl() for Salesforce Classic for Salesforce Classic
searchAndScreenPop() for Salesforce Classic for Salesforce Classic
setVisible() for Salesforce Classic for Salesforce Classic

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience
Method Parity Between Open CTI for Salesforce Classic and Lightning Experience

getPageInfo() for Salesforce Classic for Salesforce Classic

Usage

Returns information about the current page.

Syntax

```
sforce.interaction.getPageInfo(callback:function);
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
                  alert(response.result);
            } else {
                   alert(response.error);
           }
        };
       function getPageInfo() {
                //Invokes API method
                sforce.interaction.getPageInfo(callback);
        }
</script>
```

```
</head>
<body>
       <button onclick="getPageInfo();">getPageInfo</button>
</body>
</html>
```

Response

Name	Type	Description

`result`	string	Returns the URL of the current page as a JSON string, and includes any applicable object ID, object name, object type, and for API version 33.0 or later, the object tab name. For example:
		`{"url":"https://MyDomainName.my.salesforce.com/001x0000003DGQR", "objectId":"001x0000003DGQR","objectName":"Acme", "object":"Account","displayName":"Company"}`
		For API version 31.0 and later, invoking this API method on a PersonAccount object returns the following additional information.
		• accountId or contactId, the associated account or contact ID
		• personAccount, which is `true` if the object is a PersonAccount and `false` otherwise
		For example:
		`{"url":"https://MyDomainName.my.salesforce.com/001x0000003DGQR", "objectId":"001x0000003DGQR","objectName":"Acme Person Account", "object":"Account", "contactId":"003D000000QOMqg", "personAccount":true}`

`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

isInConsole() for Salesforce Classic for Salesforce Classic

Usage

Indicates if the softphone is in the Salesforce console.

Note: If this method is used in a Salesforce console where multi-monitor components is turned on, any popped out softphone components are indicated as in the console.

Syntax

```
sforce.interaction.isInConsole(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert('User is in console.');
           }
           else {
              alert('User is not in console.');
           }
        };
</script>
</head>
<body>
       <button onclick="sforce.interaction.isInConsole(callback);">isInConsole</button>
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	boolean	`true` if the softphone was in the Salesforce console, `false` if the softphone wasn't in the Salesforce console.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Salesforce Console

Salesforce Help: Turn On Multi-Monitor Components for a Salesforce Console in Salesforce Classic

isVisible() for Salesforce Classic for Salesforce Classic

Usage

Returns `true` if the softphone is visible or `false` if the softphone is hidden.

Syntax

```
sforce.interaction.isVisible(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert('Softphone is visible');
           } else {
              alert('Softphone is not visible');
           }
        };
   function isVisible() {
           sforce.interaction.isVisible(callback);
   }
</script>
</head>
<body>
       <button onclick="isVisible();">isVisible</button>
</body>
</html>
```

Response

Name	Type	Description
`result`	boolean	`true` if the softphone is visible, `false` if the softphone isn't visible.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

notifyInitializationComplete() for Salesforce Classic for Salesforce Classic

Usage

Notifies Salesforce that the softphone initialization is complete and that Salesforce should not switch to a standby URL. While the softphone initializes, a loading icon displays in the SoftPhone area. To use a standby URL, you must specify it in the call center's definition file. For more information, see Optional Call Center Elements and Attributes on page 14.

Syntax

```
sforce.interaction.cti.notifyInitializationComplete()
```

Arguments

None.

Sample Code

```
<html>
<head>
<script src="http://domain:port/support/api/29.0/interaction.js"></script>
<script type="text/javascript">
        // Invokes API method
sforce.interaction.cti.notifyInitializationComplete();
</script>
</head>
<body>
The interaction framework has been notified that the CTI initialization is complete.
</body>
</html>
```

Response

None.

onFocus() for Salesforce Classic for Salesforce Classic

Usage

Registers a function to call when the browser focus changes. In the Salesforce console, the browser focus changes when a user navigates between primary tabs or the navigation tab.

Syntax

```
sforce.interaction.onFocus( listener:function );
```

Arguments

Name	Type	Description
`listener`	function	JavaScript method called when the browser focus changes.

Sample Code–JavaScript

```
<head>
   <script type="text/javascript"
src="http//domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
     var callback = function (response) {
           if (response.result) {
                  alert(response.result);
} };
       function onFocus() {
       //Invokes API method
       sforce.interaction.onFocus(callback);
       }
</script>
</head>
<body>
       <button onclick="onFocus();">onFocus</button>
</body>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	string	Returns the URL of the page in focus as a JSON string and includes any applicable object ID, object name, object type, and for API version 33.0 or later, the object tab name. For example: `{"url":"http://MyDomainName.my.salesforce.com/001x0000003DGQR", "objectId":"001x0000003DGQR","objectName":"Acme", "object":"Account","displayName":"Company"}` If the page isn't focused on an object, the object ID, object name, and object will be empty. For API version 31.0 and later, invoking this API method on a PersonAccount object returns the following additional information. • accountId or contactId, the associated account or contact ID • personAccount, which is `true` if the object is a PersonAccount and `false` otherwise

Name	Type	Description
		For example: `{"url":"http://MyDomainName.my.salesforce.com/001x0000003DGQR", "objectId":"001x0000003DGQR","objectName":"Acme Person Account", "object":"Account", "contactId":"003D000000QOMqg", "personAccount":true}`
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Salesforce Console

Salesforce Help: Turn On Multi-Monitor Components for a Salesforce Console in Salesforce Classic

onObjectUpdate() for Salesforce Classic for Salesforce Classic

Usage

Registers a function to call when case fields, the feed, or related list data have changed on records that are displayed with a feed-based layout.

Note: Use this method with Visualforce pages you want to use as custom publishers in Case Feed.

Syntax

```
sforce.interaction.entityFeed.onObjectUpdate(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<apex:page standardController="Case">
    <apex:includeScript value="/support/api/26.0/interaction.js"/>
    <script type="text/javascript">
        var callback = function(response) {
            alert('Case was updated. Fields = ' + response.fieldsUpdated +
            ' Related lists = ' + response.relatedListsUpdated + ' Feed = ' +
              response.feedUpdated);
        };
```

```
        //Invokes API method
        sforce.interaction.entityFeed.onObjectUpdate(callback);
    </script>
</apex:page>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`fieldsUpdated`	boolean	`true` if one or more case fields were updated.
`relatedListsUpdated`	boolean	`true` if one or more case related lists were updated.
`feedUpdated`	boolean	`true` if the case feed was updated.

refreshObject() for Salesforce Classic for Salesforce Classic

Usage

Notifies a page that uses a feed-based layout, that fields, the feed, or related list data has changed, and forces an update of these on the page.

Note: Use this method with Visualforce pages you want to use as custom publishers in Case Feed.

Syntax

```
sforce.interaction.entityFeed.refreshObject(
      objectId:string,
      refreshFields:boolean,
      refreshRelatedLists:boolean,
      refreshFeed:boolean,callback:function)
```

Arguments

Name	Type	Description
`objectId`	string	The record ID of the case object.
`refreshFields`	boolean	Indicates that one or more fields on the case have changed.
`refreshRelatedLists`	boolean	Indicates that one or more case-related lists have changed.
`refreshFeed`	boolean	Indicates that the case feed has changed.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<apex:page standardController="Case">
    <apex:includeScript value="/support/api/26.0/interaction.js"/>
    <a href="javascript:void(0);"
onclick="sforce.interaction.entityFeed.refreshObject('{!case.id}', true, true, true,
function(response) {alert('Case was updated: ' + response.result);});">Refresh Case</a>
</apex:page>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	boolean	`true` if the Case Feed page was successfully updated, `false` if it was not.

refreshPage() for Salesforce Classic for Salesforce Classic

Usage

Returns `true` if page refresh is invoked, `false` otherwise. When this method is called within the Salesforce console, it refreshes the current active tab. This method is only available in API version 28.0 or later.

Syntax

```
sforce.interaction.refreshPage(callback:function);
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/28.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
                  alert('Page refresh has been invoked.');
            } else {
                   alert('Page refresh has not been invoked.');
```

```
            }
        };
       function refreshPage() {
                sforce.interaction.refreshPage(callback);
        }
</script>
</head>
<body>
       <button onclick="refreshPage();">refreshPage</button>
</body>
</html>
```

Response

Name	Type	Description
`result`	boolean	Returns `true` if page refresh has been invoked, `false` otherwise.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

refreshRelatedList() for Salesforce Classic for Salesforce Classic

Usage

Returns `true` if the related list with the given `listName` is refreshed, `false` otherwise. When this method is called within the Salesforce console, only the related list with the given list name in the currently focused view will be refreshed. This method is only available in API version 28.0 or later.

Syntax

```
sforce.interaction.refreshRelatedList(listName:string, callback:function)
```

Arguments

Name	Type	Description
`listName`	string	The name of the related list to refresh. For example, Contact for Contacts related list or Activity for Open Activities related list. Note that to refresh a custom related list created from a custom lookup field, `listName` must specify the ID of the custom lookup field.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/28.0/interaction.js"></script>
   <script type="text/javascript">
       function checkRefreshResult(result) {
           if (result.result) {
                  alert('The related list is refreshed!');
            } else {
                   alert('Cannot refresh the related list with the given listName! Make
sure the listName is correct and the related list is on the page.');
           }
        }
       function refreshActivityRelatedList() {
                sforce.interaction.refreshRelatedList('Activity', checkRefreshResult);
        }

       function refreshHistoryRelatedList() {
                sforce.interaction.refreshRelatedList('History', checkRefreshResult);
        }

       function saveAndRefresh() {
                sforce.interaction.saveLog('Task',
'Subject=ImportantTask&WhatId=[15-character ID of an account to which you want to attach
the task]', function(result) {
           if (result.result) {
                  refreshActivityRelatedList();
            } else {
                  alert('Could not save the object! Check the developer console for error
 messages.');
            }
      });
}
</script>
</head>
<body>
      <button onclick="refreshHistoryRelatedList();">Refresh History Related List</button>

       <button onclick="saveAndRefresh();">Save and Refresh</button>
</body>
</html>
```

Response

Name	Type	Description
`result`	boolean	Returns `true` if the related list with the given name is refreshed, `false` otherwise.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

Notes

- This method cannot refresh related lists created from `<apex:relatedList>`.
- This method cannot refresh a related list from an overridden Visualforce page in the Salesforce console.
- If called from within the Salesforce console, this method will only search for the related list to refresh in the currently focused view.

reloadFrame() for Salesforce Classic for Salesforce Classic

Usage

Reloads the frame that contains the page making the call. This method is available only if the record is displayed with a feed-based layout. This method is available in API version 34.0 or later.

Syntax

```
sforce.interaction.entityFeed.reloadFrame()
```

Arguments

None.

Sample Code–JavaScript

```
<apex:page standardController="Case">
    <apex:includeScript value="/support/api/34.0/interaction.js"/>
    <a href"javascript:void(0); onclick="sforce.interaction.entityFeed.reloadFrame();">
    Reload</a>
</apex:page>
```

Response

None.

runApex() for Salesforce Classic for Salesforce Classic

Usage

Executes an Apex method from an Apex class that's exposed in Salesforce.

Syntax

```
sforce.interaction.runApex(apexClass:string, methodName:string, methodParams:string,
(optional) callback:function)
```

Arguments

Name	Type	Description
`apexClass`	string	Specifies the Apex class of the method to execute.
`methodName`	string	Specifies the method to execute.
`methodParams`	string	Specifies the method parameters to pass. The string must include field value pairs and be formatted as a valid query string. For example:`name=acme&phone=(212) 555-5555`.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

1. An administrator creates an Apex class and Apex method:

```
global class AccountRetrieval{

webService static String getAccount(String name) {
   List<Account> accounts = new List<Account>();
   for (Account account : Database.query('Select Id, Name, phone from Account where Name
 like \'' + name + '%\'')){
        accounts.add(account);
   }
        String JSONString = JSON.serialize(accounts);
        return JSONString;
   }
  }
```

2. In the location where you've created the Apex class and method in Salesforce, click **Generate WSDL** to expose the method and class so that a third-party softphone can call it.
3. Add your code to the softphone:

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
        var callback = function (response) {
            if (response.result) {
                   alert(response.result);
             } else {
                    alert(response.error);
            }
         };
        function runApex() {
                  //Invokes API method
              sforce.interaction.runApex('AccountRetrieval', 'getAccount', 'name=acme',
 callback);
          }
</script>
```

```
</head>
<body>
       <button onclick="runApex();">runApex</button>
</body>
</html>
```

4. Output is returned. In this example, one account named, Acme, was found:

```
[{"attributes":{"type":"Account",
"url":"/services/data/v25.0/sobjects/Account/001x0000003DGQRAA4"},
"Id":"001x0000003DGQRAA4","Name":"Acme","Phone":"(212) 555-5555"}]
```

Response

Name	Type	Description
`result`	string	Returns the result from executing the method from the specified Apex class. No specific format is returned. The format is determined by the value from the method that was executed.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Apex Code Overview

saveLog() for Salesforce Classic for Salesforce Classic

Usage

Saves or updates an object in Salesforce.

Note: If an object uses `recordType`, pass the `recordTypeId` in the `saveLog` call. If you don't pass the `recordType`, the record is created using the default `recordType` for the profile. For person accounts, you can't pass the person account `recordType` if the profile's default is to a business account.

Syntax

```
sforce.interaction.saveLog(object:string, saveParams:string, (optional)callback:function)
```

Arguments

Name	Type	Description
`object`	string	The name of the object to save or update.

Name	Type	Description
`saveParams`	string	Specifies the fields to save or update on the object. If the object's ID is specified, a record is updated. For example: `Id=001D000000J6qIX&Name=Acme&Phone=4154561515`. If the object's ID isn't specified, a new record is created. For example: `Name=Acme&Phone=4154561515`.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
                  alert(response.result);
           } else {
                  alert(response.error);
           }
       }
       function saveLog() {
           //Invokes API method
              sforce.interaction.saveLog('Account','Name=NewAccountName&Phone=4155551212',
 callback);
       }
</script>
</head>
   <button onclick="saveLog();">saveLog</button>
</html>
```

Response

Name	Type	Description
`result`	boolean	`true` if saving or updating the object was successful, `false` if saving or updating the object wasn't successful.
`id`	string	The Id of the newly created object.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

screenPop() for Salesforce Classic for Salesforce Classic

Usage

Pops to a target URL, which must be relative.

Syntax

```
sforce.interaction.screenPop(url:string, force:boolean, (optional) callback:function)
```

Arguments

Name	Type	Description
`url`	string	A relative URL, which specifies the location of the screen pop.
`force`	boolean	Set value to `true` to force a screen pop, `false` otherwise. This argument is only available in API version 28.0 and later.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/28.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert('Screen pop was set successfully.');
           }
           else {
              alert('Screen pop failed.' + result.error);
           }
        };
       function screenPop() {
                //Invokes API method
                sforce.interaction.screenPop('/001x0000003DGQR', true, callback);
        }
</script>
</head>
<body>
       <!-- Note that '001x0000003DGQR' is an example of an object Id to screen pop. -->
       <button onclick="screenPop();">screen pop to entity Id</button>
</body>
</html>
```

Response

Name	Type	Description
`result`	boolean	`true` if the screen pop was successful, `false` if the screen pop wasn't successful.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

searchAndGetScreenPopUrl() for Salesforce Classic for Salesforce Classic

Usage

Searches objects specified in the softphone layout for a given string. Returns search results and the relative URL to be screen popped. Note that this method does not perform an actual screen pop. This method respects screen pop settings defined in the softphone layout. This method is only available in API version 28.0 or later.

Tip: This method is not available in the Open CTI API for Lightning Experience. To accomplish the same functionality in Lightning, use the `deferred` parameter available in the searchAndScreenPop() for Lightning Experience for Lightning Experience method.

Syntax

```
sforce.interaction.searchAndGetScreenPopUrl(searchParams:string, queryParams:string,
callType:string, callback:function)
```

Arguments

Name	Type	Description
`searchParams`	string	String to search.
`queryParams`	string	Specifies the query parameters to pass to the URL.
`callType`	string	Specifies the type of call, such as inbound, outbound, internal, or null. Per the settings in the softphone layout, the call type determines which objects to search for any matches. If `callType` is null, searches are inbound by default. If `callType` is internal or outbound, no screen pops occur.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/33.0/interaction.js"></script>
```

```
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert(response.result);
           } else {
              alert(response.error);
           }
        };
       function searchAndGetScreenPopUrl() {
                //Invokes API method
                sforce.interaction.searchAndGetScreenPopUrl('Acme',
'Key1=value1&Key2=value2', 'inbound', callback);
        }
</script>
</head>
<body>
       <button onclick="searchAndGetScreenPopUrl();">searchAndGetScreenPopUrl</button>
</body>
</html>
```

Response

Name	Type	Description
result	string	Returns a list of objects that match the search results and the URL to the screen pop (screenPopUrl). The search is performed on the objects specified in the softphone layout. For each object found, the object ID, field names, field values, and for API version 33.0 or later, object tab name are returned as a JSON string. The following is an example of searching for "Acme," and finding one account and three opportunity objects: {"006x0000001ZcyG":{"Name":"Acme - 600 Widgets","object":"Opportunity","displayName":"Opportunity"}, "001x0000003DGQR":{"Name":"Acme","Type":"Analyst","object":"Account", "displayName":"Company"}, "006x0000001ZcyH":{"Name":"Acme - 200 Widgets","object":"Opportunity","displayName":"Opportunity"}, "006x0000001ZcyF":{"Name":"Acme - 1,200 Widgets","object":"Opportunity","displayName":"Opportunity"}, screenPopUrl:"/search/SearchResults?searchType=2&str=Acme"} For API version 31.0 and later, invoking this API method on a PersonAccount object returns additional information: {"001D000000Jn5C5":{"Name":"PersonAccount","contactId":"003D000000QQCEu", "Type":"Analyst","object":"Account","displayName":"Account","personAccount":true}, "screenPopUrl":"/001D000000Jn5C5"}

Name	Type	Description
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Designing a Custom Softphone Layout

searchAndScreenPop() for Salesforce Classic for Salesforce Classic

Usage

Searches objects specified in the softphone layout for a given string. Returns search results and screen pops any matching records. This method respects screen pop settings defined in the softphone layout.

Note: The returned response displays only matches that meet your softphone layout settings. However, the search page that screen pops, displays all matches, regardless of the objects you specify in your sofpthone layout.

Syntax

```
sforce.interaction.searchAndScreenPop(searchParams:string, queryParams:string,
callType:string, (optional) callback:function);
```

Arguments

Name	Type	Description
`searchParams`	string	String to search.
`queryParams`	string	Specifies the query parameters to pass to the URL.
`callType`	string	Specifies the type of call, such as inbound, outbound, internal, or null. Per the settings in the softphone layout, the call type determines which objects to search for any matches. If `callType` is null, searches are inbound by default. If `callType` is internal or outbound, no screen pops occur.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/33.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
```

```
            if (response.result) {
                   alert(response.result);
             } else {
                    alert(response.error);
            }
        };
       function searchAndScreenPop() {
                //Invokes API method
                sforce.interaction.searchAndScreenPop('Acme', 'Key1=value1&Key2=value2',
'inbound', callback);
        }
</script>
</head>
<body>
       <button onclick="searchAndScreenPop();">searchAndScreenPop</button>
</body>
</html>
```

Response

Name	Type	Description
result	string	Returns a list of objects that match the search results. The search is performed on the objects specified in the softphone layout. For each object found, the object ID, field names, field values, and for API version 33.0 or later, object tab names are returned as a JSON string. Note: If multiple matches are found, only a single field is returned. The following is an example of searching for "Acme," and finding one account and three opportunity objects: `{ "006x0000001ZcyG" : {"Name" : "Acme - 600 Widgets", "object" : "Opportunity", "displayName" : "Opportunity"}, "001x0000003DGQR" : {"Name" : "Acme", "Type" : "Analyst", "object" : "Account", "displayName" : "Company"}, "006x0000001ZcyH" : {"Name" : "Acme - 200 Widgets", "object" : "Opportunity", "displayName" : "Opportunity"}, "006x0000001ZcyF" : {"Name" : "Acme - 1,200 Widgets", "object" : "Opportunity", "displayName" : "Opportunity"} }`  For API version 31.0 and later, invoking this API method on a PersonAccount object returns additional information:  `{"001D000000JWAW8":{"Name":"Acme","contactId":"003D000000QNwDB", "Type":"Analyst","object":"Account","personAccount":true}}`

Name	Type	Description
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Designing a Custom Softphone Layout

setVisible() for Salesforce Classic for Salesforce Classic

Usage

Shows or hides the softphone in the Salesforce console.

Note: If this method is used in a Salesforce console where multi-monitor components is turned on, an error will be returned.

Syntax

```
sforce.interaction.setVisible(value:boolean, (optional) callback:function)
```

Arguments

Name	Type	Description
`value`	boolean	Set value to `true` to show the softphone or set value to `false` to hide the softphone.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert(response.result);
           } else {
              alert(response.error);
           }
        };
   function setVisible(value) {
           sforce.interaction.setVisible(value, callback);
   }
</script>
```

```
</head>
<body>
       <button onclick="setVisible(false);">hide softphone</button>
</body>
</html>
```

Response

Name	Type	Description
`result`	boolean	`true` if showing or hiding the softphone succeeded, `false` if showing or hiding the softphone didn't succeed.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Salesforce Console

Salesforce Help: Turn On Multi-Monitor Components for a Salesforce Console in Salesforce Classic

Methods for Computer-Telephony Integration (CTI)

Open CTI lets you integrate your CTI system with Salesforce.

Important: The way you implement Open CTI depends on your org's user interface. There are separate Open CTI APIs for Salesforce Classic and Lightning Experience.You can't swap the two Open CTI APIs in custom JavaScript code because they behave and function differently. Make sure that you think about where you want to implement your CTI system before you begin developing.

disableClickToDial() for Salesforce Classic for Salesforce Classic

enableClickToDial() for Salesforce Classic for Salesforce Classic

getCallCenterSettings() for Salesforce Classic for Salesforce Classic

getDirectoryNumbers() for Salesforce Classic for Salesforce Classic

getSoftphoneLayout() for Salesforce Classic for Salesforce Classic

onClickToDial() for Salesforce Classic for Salesforce Classic

setSoftphoneHeight() for Salesforce Classic for Salesforce Classic

setSoftphoneWidth() for Salesforce Classic for Salesforce Classic

SEE ALSO:

Why Your UI Matters—Open CTI for Salesforce Classic vs. Lightning Experience

Method Parity Between Open CTI for Salesforce Classic and Lightning Experience

disableClickToDial() for Salesforce Classic for Salesforce Classic

Usage

Disables click-to-dial.

Syntax

```
sforce.interaction.cti.disableClickToDial( (optional) callback:function )
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert('Click to dial was disabled.');
           } else {
              alert('Click to dial was not disabled.');
           }
        };
        function disableClickToDial() {
   //Invokes API method
   sforce.interaction.cti.disableClickToDial(callback);
   }
</script>
</head>
<body>
   <button onclick="disableClickToDial();">disable click to dial</button>
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	boolean	`true` if click-to-dial was disabled, `false` if click-to-dial wasn't disabled.

Name	Type	Description
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Visualforce Developer Guide: `support:clickToDial`

enableClickToDial() for Salesforce Classic for Salesforce Classic

Usage

Enables click-to-dial.

Syntax

```
sforce.interaction.cti.enableClickToDial( (optional) callback:function )
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert('Click to dial was enabled.');
           } else {
              alert('Click to dial was not enabled.');
           }
        };
        function enableClickToDial() {
   //Invokes API method
   sforce.interaction.cti.enableClickToDial(callback);
   }
</script>
</head>
<body>
   <button onclick="enableClickToDial();">enable click to dial</button>
```

```
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	boolean	`true` if click-to-dial was enabled, `false` if click-to-dial wasn't enabled.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Visualforce Developer Guide: `support:clickToDial`

getCallCenterSettings() for Salesforce Classic for Salesforce Classic

Usage

Returns the call center settings in the call center definition file as a JSON string. For more information, see Call Center Definition Files.

Syntax

```
sforce.interaction.cti.getCallCenterSettings(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
       alert(response.result);
   }

   //Calls getCallCenterSettings
```

```
   sforce.interaction.cti.getCallCenterSettings(callback);
   </script>
</head>
<body></body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	string	If the API call was successful, the call center settings definition is returned as a JSON string. If the API call failed, null is returned.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

getDirectoryNumbers() for Salesforce Classic for Salesforce Classic

Usage

Returns the list of phone numbers from the call center's directory. This method is only available in API version 31.0 or later.

Syntax

```
sforce.interaction.cti.getDirectoryNumbers(isGlobal:boolean, callCenterName:String,
(optional) callback:function, (optional) resultSetPage:Integer, (optional)
resultSetPageSize:Integer)
```

Arguments

Name	Type	Description
`isGlobal`	boolean	Set the value to `true` to return a directory number from the global call center name, or set the value to `false` to return a directory number that is specific to a call center.
`callCenterName`	string	Specifies the call center name on which to return directory numbers. If `isGlobal` is set to `false`, and this field is not specified, all directory numbers are returned.
`callback`	function	JavaScript method executed when the API method call is completed.
`resultSetPage`	integer	Represents the page number of the list of results to return. This number starts at 0.
`resultSetPageSize`	integer	Sets the maximum number of phone numbers to retrieve, which is defaulted to 5000 and has a maximum number of 10000. If `hasNext` returns `true` in the `callback`, use this argument with `resultSetPage` to get the next page of

Name	Type	Description
		results. For example, if `resultSetPageSize` is set to 5000, and `resultSetPage` is set to 0, the first 5000 phone numbers are returned. If `resultSetPage` is set to 1, the next 5000 phone numbers are returned.

Sample Code–JavaScript

```
<html>
<head>
   <script src="https://domain:port/support/api/31.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
                  alert(response.result);
            } else {
                   alert(response.error);
           }
        };

       var isGlobal = false; //Do not return directories from the global call center
       var callCenterName = 'My Call Center'; //Call center name of directory numbers to
return

       function getDirectoryNumbers() {
                sforce.interaction.cti.getDirectoryNumbers(isGlobal, callCenterName,
callback);
        }
</script>
</head>
<body>
       <button onclick="getDirectoryNumbers();">Get Directory Numbers</button>
</body>
</html>
```

Response

Name	Type	Description
`result`	string	Returns a JSON string that represents the list of phone numbers from the specified call center name. Each phone number element contains a call center name, phone, and description. For example: `{ directoryNumbers: [ {callCenterName:'Demo Call Center', name:'Sales Cloud', phone:'415-555-1212', description:'Sales Cloud additional number'}, {callCenterName:'Demo Call Center 2', name:'Service Cloud', phone:'415-555-3434', description:'Service Cloud additional number'},`

Name	Type	Description

		<pre>],
		hasNext: false
		}</pre>

error	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

getSoftphoneLayout() for Salesforce Classic for Salesforce Classic

Usage

Returns the softphone layout as a JSON string. This method is only available in API version 27.0 or later.

Syntax

```
sforce.interaction.cti.getSoftphoneLayout(callback:function);
```

Arguments

Name	Type	Description
callback	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/27.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
                alert(response.result);
           }
       // Calls getSoftphoneLayout
       sforce.interaction.cti.getSoftphoneLayout(callback);
</script>
</head>
<body></body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description

`result`	string	If the API call was successful, the softphone layout definition is returned as a JSON string. If the API call failed, null is returned.
		
		The returned JSON string contains three elements that represent each of the call types:
		
		• `"Internal"`
		• `"Inbound"`
		• `"Outbound"`
		
		Each call-type contains three subsections:
		
		• `"callRelatedFields"`—An array of call-related fields selected to display. Possible values are `"ANI"`, `"DNIS"`, `"SEGMENT"`, and `"QUEUE"`.
		• `"objects"`—The set of Salesforce objects selected to display, along with the `Field Label` and `Field Name` (API name) selected to display from each object.
		• `"screenPopSettings"`—This object contains a `"screenPopsOpenWithin"` field with a value of either `"ExistingWindow"` or `"NewWindow"`. Additionally, it contains the settings for each of the screen pop match types: `"NoMatch"`, `"SingleMatch"`, `"MultipleMatches"`. Each match type contains a corresponding `"screenPopType"` field and may also contain a `"screenPopData"` field. If `"screenPopType"` has a value of `"PopToEntity"`, then `"screenPopData"` contains the name of the target object. If `"screenPopType"` has a value of `"PopToVisualforce"`, then `"screenPopData"` contains the name of the target Visualforcepage. If `"screenPopType"` has a value of `"PopToSearch"`, then there won't be a `"screenPopData"` field.
		
		The following is an example of a JSON response:
		
		`"Internal" : {`
		`"callRelatedFields" : [`
		`"ANI",`
		`"DNIS",`
		`]`
		`"objects" : {`
		`"User" : [ {`
		`"displayName" : "Name",`
		`"apiName" : "Name"`
		`}`
		`]`
		`},`
		`"screenPopSettings" : {}`
		`},`
		`"Inbound" : {`
		`"callRelatedFields" : [`
		`"ANI",`
		`"DNIS",`
		`"SEGMENT",`

Name	Type	Description

		<pre>"QUEUE"
		],
		"objects" : {
		"Account" : [{
		"displayName" : "Account Name",
		"apiName" : "Name"
		}
		]
		
		},
		"screenPopSettings" : {
		"NoMatch" : {
		"screenPopType" : "PopToEntity",
		"screenPopData" : "Contact"
		},
		
		"SingleMatch" : {
		"screenPopType" : "PopToVisualforce",
		"screenPopData" : "Visualforce_Page_Name"
		},
		"MultipleMatches" : {
		"screenPopType" : "PopToSearch"
		}
		}
		},
		"Outbound" : {
		"callRelatedFields" : [
		"DNIS"
		],
		"objects" : {
		"Account" : [{
		"displayName" : "Account Name",
		"apiName" : "Name"
		}
		]
		},
		"screenPopSettings" : {}
		}
		}</pre>

`error`	string or undefined	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Designing a Custom Softphone Layout

onClickToDial() for Salesforce Classic for Salesforce Classic

Usage

Registers a function to call when a user clicks an enabled phone number.

Syntax

```
sforce.interaction.cti.onClickToDial( listener:function )
```

Arguments

Name	Type	Description
`listener`	function	JavaScript method called when the user clicks a phone number.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var listener = function (response) {
           if (response.result) {
              alert('User clicked on a phone number.' + response.result );
           }
        };
   //Invokes API method
   sforce.interaction.cti.onClickToDial(listener);
</script>
</head>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	string	Returns the phone number, object ID, the name of the object, and for API version 33.0 or later, the object tab name from where the click was initiated as a JSON string. For example: `{"number":"4155551212","objectId":"001x0000003DIGj","objectName":"Account", "displayName":"Company"}`

Name	Type	Description
		For API version 33.0 or later, invoking this API method on a PersonAccount object returns the following additional information. • accountId or contactId, the associated account or contact ID • personAccount, which is `true` if the object is a PersonAccount and `false` otherwise For example: `{"number":"4155551212","objectId":"001D000000JWVvP","objectName":"Howard Jones","object":"Account","personAccount":true,"contactId":" 003D000000QOBPX"}`
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Visualforce Developer Guide: `support:clickToDial`

setSoftphoneHeight() for Salesforce Classic for Salesforce Classic

Usage

Sets the softphone height in pixels.

Note: If this method is used in a Salesforce console where multi-monitor components is turned on, an error will be returned because resizing multi-monitor component is not allowed.

Syntax

```
sforce.interaction.cti.setSoftphoneHeight(height:number, (optional) callback:function)
```

Arguments

Name	Type	Description
`height`	number	Softphone height in pixels. The height should be a number that's equal or greater than 0.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
```

```
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert('Height was set successfully.');
           }
           else {
              alert('Height was not set successfully.');
           }
        };
</script>
</head>
<body>
       <button onclick="sforce.interaction.cti.setSoftphoneHeight(200, callback);">
       set softphone height to 200px
       </button>
</body>
</html>
```

Response

Name	Type	Description
`result`	boolean	`true` if the height was set successfully, `false` if setting the height wasn't successful.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Salesforce Console

Salesforce Help: Turn On Multi-Monitor Components for a Salesforce Console in Salesforce Classic

setSoftphoneWidth() for Salesforce Classic for Salesforce Classic

Usage

Sets the softphone width in pixels for the Salesforce console.

Note: If this method is used in a Salesforce console where multi-monitor components is turned on, an error will be returned because resizing multi-monitor component is not allowed.

Syntax

```
sforce.interaction.cti.setSoftphoneWidth(width:number, (optional) callback:function)
```

Arguments

Name	Type	Description
`width`	number	Softphone width in pixels. The width should be a number that's equal or greater than 0.
`callback`	function	JavaScript method executed when the API method call is completed.

Sample Code–JavaScript

```
<html>
<head>
   <script type="text/javascript"
src="http://domain:port/support/api/25.0/interaction.js"></script>
   <script type="text/javascript">
       var callback = function (response) {
           if (response.result) {
              alert('Width was set successfully.');
           }
           else {
              alert('Width was not set successfully.');
           }
        };
</script>
</head>
<body>
       <button onclick="sforce.interaction.cti.setSoftphoneWidth(100, callback);">
       set softphone width to 100px
       </button>
</body>
</html>
```

Response

This method is asynchronous. The response is returned in an object passed to a callback method. The response object contains the following fields.

Name	Type	Description
`result`	boolean	`true` if the width was set successfully, `false` if setting the width wasn't successful.
`error`	string	If the API call was successful, this variable is undefined. If the API call failed, this variable returns an error message.

SEE ALSO:

Salesforce Help: Salesforce Console

Salesforce Help: Turn On Multi-Monitor Components for a Salesforce Console in Salesforce Classic

CHAPTER 6 Other Resources

In addition to this guide, there are other resources available for you as you learn how to use Open CTI.

Typographical Conventions Typographical Conventions
Typographical conventions are used in our code examples. Learn what Courier font, italics, and brackets mean.

SEE ALSO:

Salesforce Help: Salesforce Call Center
Salesforce Help: Salesforce Console
Salesforce Help: Glossary
Salesforce Developers: Getting Started with Salesforce Platform
Salesforce University: Training

Typographical Conventions Typographical Conventions

Typographical conventions are used in our code examples. Learn what Courier font, italics, and brackets mean.

Convention	Description
`Courier font`	In descriptions of syntax, a monospace font indicates items that you should type as shown, except for brackets. For example: `Public class HelloWorld`
`Italics`	In descriptions of syntax, italics represent variables. You supply the actual value. In the following example, three values must be supplied: *`datatype variable_name`* [= *`value`*]; If the syntax is bold and italic, the text represents a code element that needs a value supplied by you, such as a class name or variable value: `public static class YourClassHere { ... }`
`Bold Courier font`	In code samples and syntax descriptions, a bold courier font emphasizes a portion of the code or syntax.

< >	In descriptions of syntax, less-than and greater-than symbols (< >) are typed exactly as shown.
	`<apex:pageBlockTable value="{!account.Contacts}" var="contact">`
	`<apex:column value="{!contact.Name}"/>`

Convention	Description
	<pre> <apex:column value="{!contact.MailingCity}"/>\n <apex:column value="{!contact.Phone}"/>\n</apex:pageBlockTable></pre>
{ }	In descriptions of syntax, braces ({ }) are typed exactly as shown. <pre><apex:page>\n Hello {!$User.FirstName}!\n</apex:page></pre>
[ ]	In descriptions of syntax, anything included in brackets is optional. In the following example, specifying ***value*** is optional: <pre>***data_type variable_name*** [= ***value***];</pre>
|	In descriptions of syntax, the pipe sign means "or". You can do one of the following (not all). In the following example, you can create a new unpopulated set in one of two ways, or you can populate the set: <pre>Set<***data_type***> ***set_name***\n [= new Set<***data_type***>();] |\n [= new Set<***data_type***{***value*** [, ***value2***. . .] };] |\n ;</pre>