Salesforce Console Developer Guide

Version 63.0, Spring ’25

Last updated: January 3, 2025

CONTENTS

CHAPTER 1 Salesforce Console Developer Guide

The Lightning Console JavaScript API and the Salesforce Console Integration Toolkit both interact with Salesforce console apps. This guide provides reference material for both.

Starting with API version 42.0 of the Salesforce Console Integration Toolkit, many of the methods used in existing Visualforce pages and third-party web tabs now work in Lightning Experience. Just point to the latest version of the toolkit script in your Visualforce pages or third-party web tabs. Third-party content must be allowlisted in the Trusted URLs list to be used in Lightning Experience. See Classic Console API Methods Supported in the Lightning Console API for a list of supported methods.

To use this guide, it helps if you have a basic familiarity with:

- JavaScript
- Visualforce
- Web services
- Software development
- Salesforce console
- Lightning
- Lightning console apps

Note: As of Spring '19 (API version 45.0), you can build Lightning components using two programming models: the Lightning Web Components model, and the original Aura Components model. Lightning web components are custom HTML elements built using HTML and modern JavaScript. Lightning web components and Aura components can coexist and interoperate on a page. This developer guide covers Aura components only.

IN THIS SECTION:

Get to Know Lightning Console

Get started with the Salesforce console in Lightning Experience.

Get to Know the Utility Bar

The utility bar is a specialized type of Lightning page that gives your users quick access to common productivity tools. Utility bars are supported in Lightning Experience for desktop only.

SEE ALSO:

How are the Classic and Lightning Console APIs Different?

Lightning Console JavaScript API

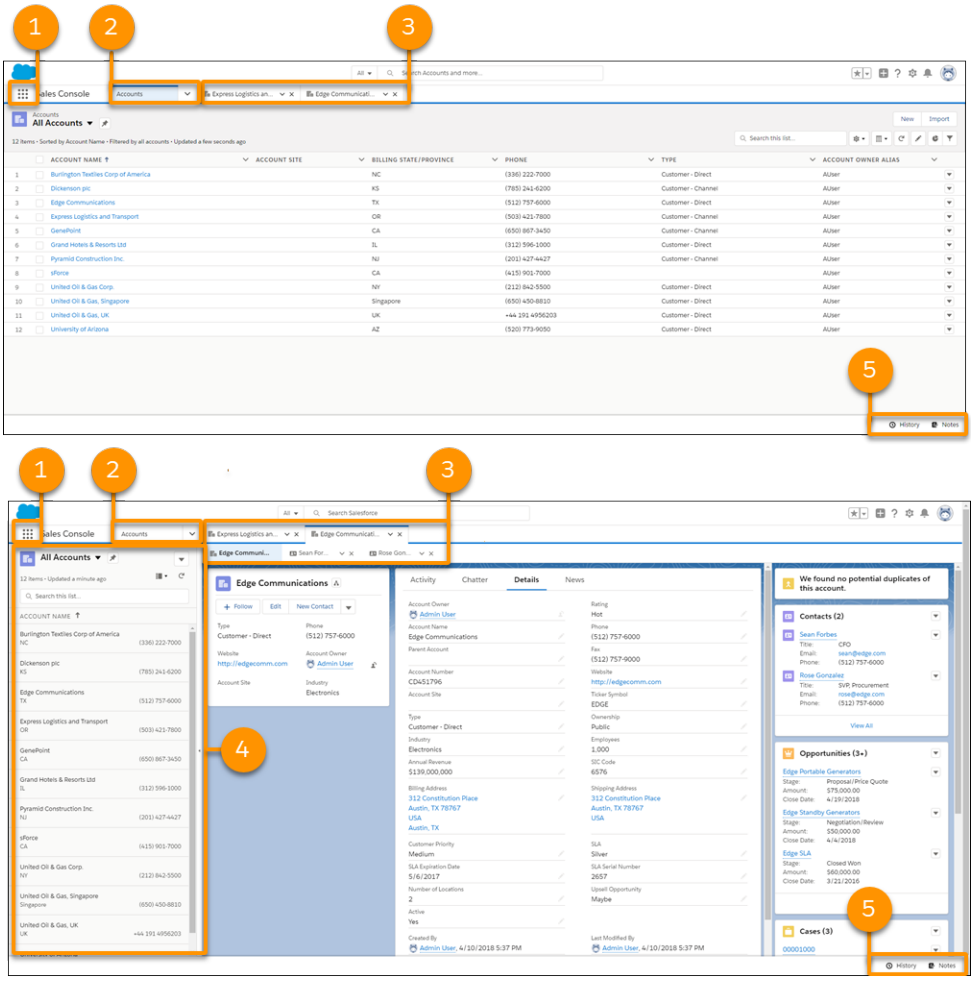
Salesforce Console Integration Toolkit for Salesforce Classic

Get to Know Lightning Console

Get started with the Salesforce console in Lightning Experience.

Use workspace API methods from Lightning pages either in the utility bar or in a Lightning console app. Here's how a Lightning console app works:

Lightning Console App User Interface



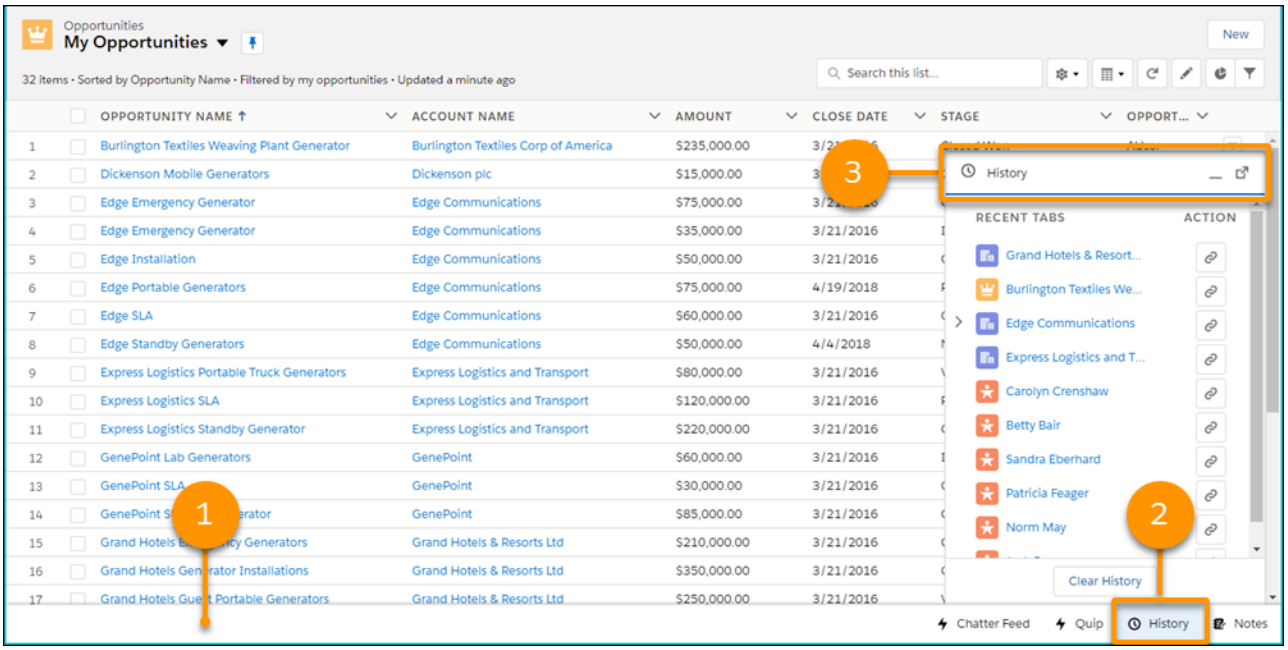
- The App Launcher (1) lets you switch between apps. To switch to another console app or back to a standard app, use the App Launcher. The name of the app you're currently in is displayed next to the App Launcher.
- The navigation menu (2) displays the navigation item you currently have selected. To open the navigation menu, click . From there, you can view or edit your navigation items. Selecting a navigation item opens the navigation item's home page. Objects open in table view. Opening a record changes the view to split view. Once in split view, click the navigation item again to switch back to table view, or use the **Display as** dropdown.
- Records open in workspace tabs, and related records opened from inside a workspace tab open in subtabs (3). You can refresh, pin, customize, and close a tab using the tab menu. You can also open navigation items in a new workspace tab by using **Ctrl+click** or **Cmd+click**.
- The split view panel (4) can be hidden with. Records opened from the split view panel open in new workspace tabs.
- The utility bar (5) lets you access common processes and tools like History and Notes.

Get to Know the Utility Bar

The utility bar is a specialized type of Lightning page that gives your users quick access to common productivity tools. Utility bars are supported in Lightning Experience for desktop only.

A utility is broadly defined as a single-column Lightning page. Salesforce provides you with several ready-to-use utilities, such as Recent Items, History, and Notes. You can also make your own, and customize the utility bar in Setup. From Setup, enter `App Manager` in the Quick Find box, then select **App Manager**. Either click **New Lightning App** to create an app, or click **Edit** next to an existing Lightning app to add a utility bar or edit the existing one. The utility bar API includes a set of methods for working with utilities and the utility bar.

To add a utility bar, add at least one utility item that isn't a background utility item. To remove a utility bar, remove all non-background utility items from your app.



1. The utility bar. This utility bar includes four utilities: Chatter Feed, Quip, History, and Notes. Each utility has an icon and label.
2. The selected utility. The selected utility opens in a panel.
3. The panel header, showing the panel icon and label.

SEE ALSO:

Salesforce Help: Add a Utility Bar to Lightning Apps

Methods for the Utility Bar in Lightning Experience

Using Background Utility Items

CHAPTER 2 Lightning Console API

Lightning console apps allow users to quickly find the information they need, and make edits while viewing multiple records on one screen. The Lightning Console JavaScript API gives you programmatic access to Lightning console apps, so you can fully integrate Lightning console apps with Aura components and Lightning web components while extending them to meet your business needs.

The Lightning Console JavaScript API includes three libraries, the navigation item API, the utility bar API, and the workspace API.

- The navigation item API provides methods that can be used from Aura components to interact with the console's navigation menu. This API is used in Lightning console apps only.
- The utility bar API provides methods that can be used from Aura components and Lightning web components in the utility bar to open, resize, or minimize a utility. This API is used in Lightning apps with standard or console navigation.
- The workspace API provides methods for Aura components and Lightning web components for opening, closing, and getting information about workspace tabs and subtabs. This API is used in Lightning console apps only.

For a full list of methods in each API, see Methods for Lightning Console JavaScript API.

You can build Lightning components using two programming models: the Lightning Web Components model and Aura Components model. Although both models can coexist and interoperate on a page, we recommend that you build your apps with Lightning Web Components, which use HTML and modern JavaScript.

EDITIONS

Available in: Lightning Experience

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

Lightning console apps are available for an extra cost to users with Salesforce Platform user licenses for certain products. Some restrictions apply. For pricing details, contact your Salesforce account executive.

Note: Only the utility bar and workspace APIs are currently supported for Lightning web components.

IN THIS SECTION:

How are the Classic and Lightning Console APIs Different?

The user interface of your org dictates which development tools you can use with the Salesforce console.

Console Integration Toolkit versus Lightning Console JavaScript API

Both the Lightning Console JavaScript API and the Salesforce Console Integration Toolkit are JavaScript APIs that allow you to interact with Classic or Lightning console apps. Methods are implemented differently in each API, however.

Here's what's different between the Lightning Console JavaScript API and the Salesforce Console Integration Toolkit.

You Use the Methods in Different Places

- In Aura components, use the Lightning Console JavaScript API methods in the JavaScript controller of a Lightning component.
- In Lightning web components, you can use only the utility bar and workspace API methods, wire adapters, and Lightning message channels. Lightning web components don't currently support working with navigation items.
- Visualforce or iframed, third-party pages work in both Lightning Experience and Salesforce Classic. For Visualforce and iframe pages, use the Classic methods from the Salesforce Console Integration Toolkit. However, there are limitations regarding which methods you can use. Classic Console API Methods Supported in the Lightning Console API, provides details on the supported methods.

  When you are using the Salesforce Console Integration Toolkit in Salesforce Classic, you use methods within `<script>` tags for Visualforce pages or iframed, third-party pages.

The Input Syntax for Methods is Different

Methods in the Lightning Console JavaScript API (Aura components) take a JSON array of arguments:

```
workspace.openTab({
    url: '#https://salesforce.com',
    focus: true,
    label: 'Salesforce',
});
```

Similarly, methods in the Lightning Console JavaScript API (LWC) take a JSON array of arguments:

```
openTab({
    url: '#/sObject/001R0000003HgssIAC/view',
    label: 'Global Media',
    focus: true
})
```

Note: For Aura components, required parameters are passed to the method in an object. For LWC, required parameters are explicitly passed to the method.

Methods in the Salesforce Console Integration Toolkit don't:

```
sforce.console.openPrimaryTab(null, 'https://salesforce.com', false,
                'salesforce', openSuccess, 'salesforceTab');
```

The APIs Provide Different Methods

Although some of the methods in the Lightning Experience methods are similar to the Salesforce Classic methods, they have different names and provide different functionality.

The Lightning Console JavaScript API also provides methods for use with the utility bar, which is available in Lightning Experience only.

IN THIS SECTION:

Console API Method Parity—What's Different Between Lightning Experience and Salesforce Classic?
The Lightning Console JavaScript API provides methods similar to those methods in the Salesforce Console Integration Toolkit.

Classic Console API Methods Supported in the Lightning Console API

Visualforce pages and third-party web tabs that use some Salesforce Console Integration Toolkit methods work in Lightning Experience as-is. Just point to the latest version of the toolkit script in your Visualforce pages or third-party web tabs. Third-party content must be allowlisted with CSP directives via Trusted URLs. This table lists the Salesforce Console Integration Toolkit methods that you can use in Lightning Console JavaScript API starting with API version 42.0.

Utility Bar API Method Parity

The utility bar API provides methods for Aura Components and Lightning Web Components (LWC) in Lightning Experience only. Salesforce Classic isn't supported.

SEE ALSO:

Lightning Console JavaScript API

Salesforce Console Integration Toolkit for Salesforce Classic

Console API Method Parity—What's Different Between Lightning Experience and Salesforce Classic?

The Lightning Console JavaScript API provides methods similar to those methods in the Salesforce Console Integration Toolkit.

This table shows which Salesforce Console Integration Toolkit (Salesforce Classic) methods map to Lightning Console JavaScript API (Lightning Experience) methods and events. Not every Salesforce Console Integration Toolkit has a Lightning analog. You can replicate some Classic methods using Lightning events, combining Lightning Experience methods, or using iterative and conditional logic with methods and events.

Important: Only Salesforce Console Integration Toolkit methods with a Lightning Console JavaScript API or workaround appear in this table. Methods without alternatives or workarounds are not listed.

For Lightning Experience, you can build your apps using Lightning Web Components (LWC) and Aura components. Both can interoperate on the same page.

Methods for Primary Tabs and Subtabs

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Aura Components for Lightning Experience)	LWC Support
closeTab()	closeTab()	Same as Aura Components.
focusPrimaryTabById()	focusTab()	Same as Aura Components.
focusSubtabById()	focusTab()	Same as Aura Components.
getEnclosingPrimaryTabId()	Use the Lightning method `getEnclosingTabId()`. If the calling component is within a subtab, then the subtab ID is returned. If the calling component is within a workspace tab, then the workspace ID is returned.	Use the `EnclosingTabId` wire adapter.
getEnclosingPrimaryTabObjectId()	Use `getEnclosingTabId()` to get the tab ID. Then, use the tab ID to call getTabInfo(tabId), which includes the object ID in the response payload (if applicable).	Use the `EnclosingTabId` wire adapter.

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Aura Components for Lightning Experience)	LWC Support
getEnclosingTabId()	Use the Lightning method `getEnclosingTabId()`. If the calling component is within a subtab, then the subtab ID is returned. If the calling component is within a workspace tab, then the workspace ID is returned.	Use the `EnclosingTabId` wire adapter.
getFocusedPrimaryTabId()	getFocusedTabInfo()	Same as Aura Components.
getFocusedPrimaryTabObjectId()	getFocusedTabInfo()	Same as Aura Components.
getFocusedSubtabId()	getFocusedTabInfo()	Same as Aura Components.
getPageInfo()	getTabInfo()	Same as Aura Components.
getPrimaryTabIds()	Not supported. **Workaround:** Call `getAllTabInfo()`.	Same as Aura Components.
getSubtabIds()	Not supported. **Workaround:** Call `getAllTabInfo()` to get a list of all workspace tab objects. Iterate through each workspace tab object, collecting subtab IDs where applicable	Same as Aura Components.
getTabLink()	getTabURL()	Use getTabInfo().
onEnclosingTabRefresh()	Use lightning:tabRefreshed with `getEnclosingTabId()`.	Use the `lightning__tabRefreshed` Lightning message channel.
onFocusedPrimaryTab()	lightning:tabFocused	Use the `lightning__tabFocused` Lightning message channel.
onFocusedSubtab()	lightning:tabFocused	Use the `lightning__tabFocused` Lightning message channel.
onTabSave()	Not supported.	Not supported.
openConsoleUrl()	openConsoleUrl()	Not supported.
openPrimaryTab()	openTab()	Same as Aura Components.
openSubtab()	openSubtab()	Same as Aura Components.
refreshPrimaryTabById()	Use refreshTab() and specify a workspace tab ID. The `activate` argument isn't supported in the Lightning API. Use `refreshTab()` with focusTab(). instead.	Same as Aura Components.

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Aura Components for Lightning Experience)	LWC Support
refreshSubtabById()	Use refreshTab() and specify a subtab ID. The `activate` argument isn't supported in the Lightning API. Use `refreshTab()` with focusTab(). instead.	Same as Aura Components.
setTabIcon()	setTabIcon()	Same as Aura Components.
setTabTitle()	setTabLabel()	Same as Aura Components.
setTabUnsavedChanges()	Use the lightning:unsavedChanges component.	Not supported.

Methods for Navigation Tabs

The `force:navigateToObjectHome` Lightning event allows you to complete actions for many navigation tab methods in Salesforce Classic.

Note: LWC doesn't currently support working with navigation tabs.

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Lightning Experience)
focusNavigationTab()	focusNavigationItem()
getNavigationTabs()	getNavigationItems()
getSelectedNavigationTab()	getSelectedNavigationItem()
refreshNavigationTab()	refreshNavigationItem()
setSelectedNavigationTab()	force:navigateToObjectHome

Methods for Application-Level Custom Console Components

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Aura Components for Lightning Experience)	LWC Support
blinkCustomConsoleComponentButtonText()	Not supported. **Workaround:** Use setUtilityLabel.	updateUtility()
isCustomConsoleComponentWindowHidden()	getUtilityInfo()	getInfo()
onCustomConsoleComponentButtonClicked()	onUtilityClick()	Same as Aura Components.
setCustomConsoleComponentButtonIconUrl()	setUtilityIcon() setPanelHeaderIcon()	updateUtility()
setCustomConsoleComponentButtonStyle()	setUtilityHighlighted	updateUtility()

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Aura Components for Lightning Experience)	LWC Support
setCustomConsoleComponentButtonText()	setUtilityLabel	updateUtility()
setCustomConsoleComponentHeight()	setPanelHeight()	updatePanel()
setCustomConsoleComponentVisible()	openUtility() minimizeUtility()	minimize()
setCustomConsoleComponentWidth()	setPanelWidth()	updatePanel()

Methods for Live Agent

Note: LWC doesn't currently support working with Live Agent.

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Lightning Experience)
endChat()	endChat()
getChatLog()	getChatLog()
sendCustomEvent()	sendCustomEvent()
sendMessage()	sendMessage()

Methods for Omni-Channel

Note: LWC doesn't currently support working with Omni-Channel.

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Lightning Experience)
acceptAgentWork()	acceptAgentWork()
closeAgentWork()	closeAgentWork()
declineAgentWork()	declineAgentWork()
getAgentWorkload()	getAgentWorkload()
getAgentWorks()	getAgentWorks()
getServicePresenceStatusChannels()	getServicePresenceStatusChannels()
getServicePresenceStatusId()	getServicePresenceStatusId
login()	login()
logout()	logout()
sertServicePresenceStatus()	setServicePresenceStatus()

Methods for Computer-Telephony Integration (CTI)

Note: LWC doesn't currently support working with CTI.

Salesforce Console Integration Toolkit (Salesforce Classic)	Lightning Console JavaScript API Method (Lightning Experience)
onCallBegin()	Not supported for both Aura and LWC.
onCallEnd()	Not supported for both Aura and LWC.
onCallLogSaved()	Not supported for both Aura and LWC..

Classic Console API Methods Supported in the Lightning Console API

Visualforce pages and third-party web tabs that use some Salesforce Console Integration Toolkit methods work in Lightning Experience as-is. Just point to the latest version of the toolkit script in your Visualforce pages or third-party web tabs. Third-party content must be allowlisted with CSP directives via Trusted URLs. This table lists the Salesforce Console Integration Toolkit methods that you can use in Lightning Console JavaScript API starting with API version 42.0.

Salesforce Console Integration Toolkit methods that aren't supported in Lightning Experience result in a failure error message.

Important: Only API versions 42.0 and above of the Salesforce Console Integration Toolkit are supported in the Lightning Console JavaScript API. Only API versions 43.0 and above are supported in Open CTI.

Methods for Primary Tabs and Subtabs

Note: Methods using `objectId` return 18-character, case-insensitive record IDs when invoked from within a Lightning console. When invoked from within a Salesforce Classic console, they return 15-character, case-sensitive record IDs.

Workspace tab and subtab IDs in a Lightning console use a different format from Salesforce Classic console primary tab and subtab IDs. Any code that validates the format of tab IDs must be updated or removed to account for the change. A Salesforce Classic console tab ID can look like `scc-pt-1` or `scc-st-1`. A Lightning console tab ID looks like `ctab1` or `ctab1_3`.

Salesforce Classic Method	Supported in Lightning Console (Aura)	Supported in Lightning Console (LWC)	Notes About Use in Lightning Console
`closeTab()`	✔	✔	
`disableTabClose()`	✔	✔	
`focusPrimaryTabById()`	✔	✔	
`focusPrimaryTabByName()`	✖	✖	
`focusSidebarComponent()`	✖	✖	
`focusSubtabById()`	✔	✔	
`focusSubtabByNameAndPrimaryTabId()`	✖	✖	

Salesforce Classic Method	Supported in Lightning Console (Aura)	Supported in Lightning Console (LWC)	Notes About Use in Lightning Console
focusSubtabByNameAndPrimaryTabName()	✗	✗	
generateConsoleUrl()	✗	✗	
getEnclosingPrimaryTabId()	✓	✗	
getEnclosingTabId()	✓	✓	
getFocusedPrimaryTabId()	✓	✓	
getFocusedPrimaryTabObjectId()	✓	✓	
getFocusedSubtabId()	✓	✓	
getFocusedSubtabObjectId()	✓	✗	
getPageInfo()	See notes	✗	These fields aren't supported and aren't returned in the response: • object • displayName • accountId • contactId • personAccount
getPrimaryTabIds()	✓	✓	
getSubtabIds()	✓	✗	
getTabLink()	See notes	✗	The level argument `sforce.console.TabLink.PARENT_AND_CHILDREN` isn't supported.
isInConsole()	✓	✓	
onEnclosingTabRefresh()	✓	✗	
onFocusedPrimaryTab()	✓	✗	
onFocusedSubtab()	✓	✗	Utility items aren't supported in the Lightning API.
onTabSave()	✗	✗	
openConsoleUrl()	✗	✗	
openPrimaryTab()	See notes	Make sure to add third-party domains to the Trusted URLs list.	

Salesforce Classic Method	Supported in Lightning Console (Aura)	Supported in Lightning Console (LWC)	Notes About Use in Lightning Console
		The following aren't supported in the Lightning API: • `id` argument • `name` argument. As an alternative, save the `tabId` that's returned and use it in your API calls.	
`openSubtab()`	See notes	Make sure to add third-party domains to the Trusted URLs list. The following aren't supported in the Lightning API: • `id` argument • `name` argument. As an alternative, save the `tabId` that's returned and use it in your API calls.	
`openSubtabByPrimaryTabName()`	✗	✓	
`refreshPrimaryTabById()`	See notes	See notes	The `fullRefresh` argument isn't supported in the Lightning API.
`refreshPrimaryTabByName()`	✗	✓	
`refreshSubtabById()`	See notes	See notes	The `fullRefresh` argument isn't supported in the Lightning API.
`refreshSubtabByNameAndPrimaryTabId()`	✗	✗	
`refreshSubtabByNameAndPrimaryTabName()`	✗	✗	
`reopenLastClosedTab()`	✗	✗	
`resetSessionTimeOut()`	✗	✗	
`setTabUnsavedChanges()`	✗	✗	

Salesforce Classic Method	Supported in Lightning Console (Aura)	Supported in Lightning Console (LWC)	Notes About Use in Lightning Console
`setTabIcon()`	See notes	See notes	Only Salesforce Lightning Design System icons are supported for `iconUrl`. URLs and custom icons aren't supported. Sample supported values: • `sforce.console.setTabIcon("standard:email")` • `sforce.console.setTabIcon("action:new")` • `sforce.console.setTabIcon("custom:custom1")`
`setTabLink()`	✓	✗	
`setTabStyle()`	✗	✗	
`setTabTextStyle()`	✗	✗	
`setTabTitle()`	✓	✓	

Methods for Application-Level Custom Console Components

Note: LWC doesn't currently support working with these methods.

The following methods must be called from within a Lightning utility.

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`addToBrowserTitleQueue()`	✓	
`blinkCustomConsoleComponentButtonText()`	✗	
`isCustomConsoleComponentPoppedOut()`	✓	
`isCustomConsoleComponentHidden()`	✓	
`isInCustomConsoleComponent()`	✓	
`onCustomConsoleComponentButtonClicked()`	✓	
`removeFromBrowserTitleQueue()`	✓	
`runSelectedMacro()`	✗	
`scrollCustomConsoleComponentButtonText()`	✗	
`selectMacro()`	✗	

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`setCustomConsoleComponentButtonIconUrl()`	See notes	In Lightning Console, URL values for icons aren't supported in utility bar utilities. Only Salesforce Lightning Design System are supported. Sample supported `iconUrl` values: • `setCustomConsoleComponentButtonIconUrl("clock");` • `setCustomConsoleComponentButtonIconUrl("utility:clock");`
`setCustomConsoleComponentButtonStyle()`	✗	
`setCustomConsoleComponentButtonText()`	✓	
`setCustomConsoleComponentHeight()`	✓	
`setCustomConsoleComponentVisible()`	✓	
`setCustomConsoleComponentWidth()`	✓	
`setCustomConsoleComponentPopoutable()`	✓	
`setCustomConsoleComponentWindowVisible()`	✗	
`setSidebarVisible()`	✗	

Methods for Navigation Tabs

Note: LWC doesn't currently support working with these methods.

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`focusNavigationTab()`	✓	
`getNavigationTabs()`	✓	
`getSelectedNavigationTab()`	✓	
`refreshNavigationTab()`	✓	
`setSelectedNavigationTab()`	✗	

Methods for Live Agent

Note: LWC doesn't currently support working with these methods.

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`acceptChat()`	✗	

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
cancelFileTransferByAgent()	✗	
declineChat()	✓	
endChat()	✓	
getAgentInput()	✗	
getAgentState()	✗	
getChatLog()	✓	
getChatRequests()	✗	
getDetailsByChatKey()	✗	
getDetailsByPrimaryTabId()	✗	
getEngagedChats()	✗	
getMaxCapacity()	✗	
initFileTransfer()	✗	
onAgentSend()	✗	
onAgentStateChanged()	✗	
onChatCanceled()	✗	
onChatCriticalWaitState()	✗	
onChatDeclined()	✗	
onChatEnded()	✗	
onChatRequested()	✗	
onChatStarted()	✗	
onChatTransferredOut()	✗	
onCurrentCapacityChanged()	✗	
onCustomEvent()	✗	
onFileTransferCompleted()	✗	
onNewMessage()	✗	
onTypingUpdate()	✗	
sendCustomEvent()	✓	
sendMessage()	✓	
setAgentInput()	✗	

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`setAgentState()`	✖	

Methods for Omni-Channel

Note: LWC doesn't currently support working with these methods.

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`acceptAgentWork()`	✔	
`closeAgentWork()`	✔	
`declineAgentWork()`	✔	
`getAgentWorkload()`	✔	
`getAgentWorks()`	✔	
`getServicePresenceStatusChannels()`	✔	
`getServicePresenceStatusId()`	✔	
`login()`	✔	
`logout()`	✔	
`setServicePresenceStatus()`	✔	

Methods for Console Events

Note: LWC doesn't currently support working with these methods.

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`addEventListener()`	✔	`sforce.console.ConsoleEvent.CONSOLE_LOGOUT` isn't supported in the Lightning API. `sforce.console.ConsoleEvent.CLOSE_TAB` returns the ID of the closed tab only. The Lightning API doesn't return the objectId or the tabObjectId. The Lightning API doesn't return special message responses from custom keyboard shortcuts. However, if the response is from a console event, the message includes payload details.

Salesforce Classic Method	Supported in Lightning Console	Notes About Use in Lightning Console
`fireEvent()`	See notes	`fireEvent()` returns `success true` even when eventListeners for the given eventType are removed.
`removeEventListener()`	✔	

Utility Bar API Method Parity

The utility bar API provides methods for Aura Components and Lightning Web Components (LWC) in Lightning Experience only. Salesforce Classic isn't supported.

This table shows how the LWC methods map to Aura Components methods. Both can interoperate on the same page. We recommend using LWC to build user interfaces using modern web standards.

Important: Before you can use the utility bar API with LWC, Lightning Web Security must be enabled in your organization.

Methods for Utility Bars

Utility Bar Methods for LWC	Utility Bar Methods for Aura Components
enableModal()	toggleModalMode()
enablePopout()	disableUtilityPopOut()
getAllUtilityInfo()	Same as LWC.
getInfo()	getUtilityInfo()
minimize()	minimizeUtility()
onUtilityClick()	Same as LWC.
open()	openUtility()
updatePanel()	Update a utility panel using one or more of these methods. • setPanelHeaderIcon() • setPanelHeaderLabel() • setPanelHeight() • setPanelWidth()
updateUtility()	Update a utility using one or more of these methods. • setUtilityHighlighted() • setUtilityIcon() • setUtilityLabel()
EnclosingUtilityId context wire adapter	getEnclosingUtilityId()

Utility Bar API Considerations

Consider these additional guidelines.

- For LWC, the `utilityId` parameter is always required. For Aura Components, `utilityId` can be optional if it's called from within a utility.
- For LWC, to retrieve the record context from a component in a utility, use the `CurrentPageReference` wire adapter on page 30. For Aura Components, implement the `force:hasRecordId` interface on your custom component.
- The isUtilityPoppedOut() Aura Components method doesn't have an LWC equivalent. You can use the getAllUtilityInfo() method to retrieve the same information.

Lightning Console JavaScript API

Lightning console apps allow users to quickly find the information they need, and make edits while viewing multiple records on one screen. The Lightning Console JavaScript API gives you programmatic access to Lightning console apps, so you can fully integrate Lightning console apps with Aura components and Lightning web components while extending them to meet your business needs.

The Lightning Console JavaScript API includes three libraries: the navigation item API, the utility bar API, and the workspace API.

- The navigation item API provides methods that can be used from Aura components to interact with the console's navigation menu. This API is used in Lightning console apps only.
- The utility bar API provides methods that can be used from Aura components and Lightning web components in the utility bar to open, resize, or minimize a utility. This API is used in Lightning apps with standard or console navigation.
- The workspace API provides methods for Aura components and Lightning web components for opening, closing, and getting information about workspace tabs and subtabs. This API is used in Lightning console apps only.

For a full list of methods in each API, see Methods for Lightning Console JavaScript API.

EDITIONS

Available in: Lightning Experience

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

Lightning console apps are available for an extra cost to users with Salesforce Platform user licenses for certain products. Some restrictions apply. For pricing details, contact your Salesforce account executive.

You can build Lightning components using two programming models: the Lightning Web Components model and Aura Components model. Although both models can coexist and interoperate on a page, we recommend that you build your apps with Lightning Web Components, which use HTML and modern JavaScript.

Note: Only the utility bar and workspace APIs are currently supported for Lightning web components.

IN THIS SECTION:

Lightning Console JavaScript API Syntax

Use Lightning Console JavaScript API methods in the JavaScript file of a Lightning web component or in the JavaScript controller of an Aura component.

Using Background Utility Items

Implement the `lightning:backgroundUtilityItem` interface to create a component that fires and responds to events without rendering in the utility bar.

Using Pop-Out Utilities

Utilities that support pop-out can be "popped out" of the utility bar and into their own separate child windows.

Using Events with the Lightning Console JavaScript API

The Lightning framework uses event-driven programming, which allows you to create handlers to respond to interface events as they occur. The Lightning Console JavaScript API provides several events specific to Lightning console apps.

Using Page Context in the Utility Bar API

In both Lightning console apps and standard navigation apps, utilities can respond to the context of the current page. For a Lightning web component, use the `CurrentPageReference` wire adapter. For an Aura component, specify `implements="force:hasRecordId"` to access the `recordId` of the record a user is viewing.

Using Page References to Open Console Workspace Tabs and Subtabs

You can navigate to different page types, including a URL addressable custom component. To make a custom component URL addressable using LWC, use the `lightning__UrlAddressable` target. To make an Aura component URL addressable, implement the `lightning:isUrlAddressable` interface on your custom component.

Debugging

Use your browser's console and JavaScript error messages generated within Salesforce to debug Lightning pages built with the Lightning Console JavaScript API. The methods in the Lightning Console JavaScript APIs are asynchronous and return their results using promises.

Methods for Lightning Console JavaScript API

If your org is using Lightning Experience, use Lightning Console JavaScript API methods.

Events for Lightning Console JavaScript API

Use events and handlers in your Aura components and controllers to respond to events like workspace tabs opening, closing, or gaining focus. In Lightning web components, subscribe to Aura application events using their corresponding Lightning message channels.

SEE ALSO:

Methods for Lightning Console JavaScript API

Lightning Console JavaScript API Syntax

Use Lightning Console JavaScript API methods in the JavaScript file of a Lightning web component or in the JavaScript controller of an Aura component.

LWC Syntax

To use LWC Workspace API, import `lightning/platformWorkspaceApi` in your JavaScript code.

The `lightning/platformWorkspaceApi` module gives you access to workspace API methods, wire adapters, and Lightning message channels. Access Lightning message channels on page 141 by importing from `@salesforce/messageChannel/lightning__tab*`. For example, `@salesforce/messageChannel/lightning__tabClosed`.

The following example shows a Lightning web component that uses the `openSubtab` API method on page 63 and `EnclosingTabId` wire adapter on page 53.

```
import { LightningElement, wire } from 'lwc';
import { EnclosingTabId, openSubtab } from 'lightning/platformWorkspaceApi';

export default class MyComponent extends LightningElement {
  @wire(EnclosingTabId) tabId;
```

```
  handleClick() {
    if (!this.tabId) {
      return;
    }
    // Open a record as a subtab of the current tab
    openSubtab(this.tabId, { recordId: 'YourRecordId', focus: true });
  }
}
```

Configure the component's `.js-meta.xml` file so the component can be accessed in the Lightning App Builder.

```
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
  <apiVersion>59.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
      <target>lightning__RecordPage</target>
      <target>lightning__AppPage</target>
      <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>
```

LWC supports Workspace API methods only. Similar to the Aura counterpart, methods in the Workspace API take a JSON object as an argument. The values included in the object depend on the method. For example, `openTab` takes an object that includes the `url` and `focus` (whether the new tab has focus). Check the reference section of this guide before using a method so that you know which arguments to pass to it.

Example: The lwc-recipes repo contains many LWC Workspace API examples. Look for components that start with `workspaceApi`, for example, workspaceAPICloseTab

Aura Components Syntax

To use the Lightning Console JavaScript API, include `lightning:navigationItemAPI`, `lightning:workspaceAPI`, or `lightning:utilityBarAPI` in your Aura component.

The `lightning:navigationItemAPI`, `lightning:workspaceAPI`, and `lightning:utilityBarAPI` components give you access to their coordinating APIs. Give each component an `aura:id` so that you can reference it from the component's controller.

The following example shows a simple Aura component that uses the API libraries:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:navigationItemAPI aura:id="navigationItem" />
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:utilityBarAPI aura:id="utilityBar" />
    <lightning:button label="Focus Navigation Item" onclick="{!c.focusNavigationItem }"
/>
    <lightning:button label="Open Utility" onclick="{!c.openUtilityBar }"/>
    <lightning:button label="Open Tab" onclick="{!c.openTab }" />
</aura:component>
```

This component implements `flexipage:availableForAllPageTypes` so that it can be accessed in the Lightning App Builder.

The component's JavaScript controller looks like this.

```
({
    openUtilityBar : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.openUtility();
    },

    openTab: function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            pageReference: {
                "type": "standard__recordPage",
                "attributes": {
                    "recordId":"500xx000000Ykt2AAC",
                    "actionName":"view"
                },
                "state": {}
            },
            focus: true
        }).then(function(response) {
            workspaceAPI.getTabInfo({
            tabId: response
        }).then(function(tabInfo) {
            console.log("The recordId for this tab is: " + tabInfo.recordId);
        });
        }).catch(function(error) {
            console.log(error);
        });
    },

    focusNavigationItem : function(component, event, helper) {
        var navigationItemAPI = component.find("navigationItem");
        navigationItemAPI.focusNavigationItem().then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

The controller has three functions, each of which uses an API method. To use a method in a controller, use `component.find` with the `aura:id` you gave to the `lightning:navigationItemAPI`, `lightning:workspaceAPI`, or `lightning:utilityBarAPI`.

Methods in the Workspace API and the Utility Bar API take a JSON object as an argument. The values included in the object depend on the method. `openTab`, for example, takes an object that includes the `url` and `focus` (whether the new tab has focus). Check the reference section of this guide before using a method so that you know which arguments to pass to it.

LWC VS Aura Guidelines

When working with the Lightning Console JavaScript API, consider these guidelines.

- In LWC, required parameters are explicitly passed to the method like `focusTab(tabId);`. In Aura, required parameters are passed to the method in an object like `workspaceAPI.focusTab({tabId : response});`.
- In LWC, pass in a URL that matches a Lightning Experience page, for example, `/lightning/r/Account/001R0000003HgssIAC/view`

IN THIS SECTION:

JavaScript Promises

Methods in the Lightning Console JavaScript API return results using promises.

Note: Examples in this guide don't include the `$A.getCallback()` wrapper because the Lightning Console JavaScript API returns promises that already include the `$A.getCallback()` wrapper around callback functions. This is reflected in the sample code throughout this guide.

Use JavaScript Promises in LWC

This example uses the `openTab()` to get the tab ID of the focused tab. Then the function calls `focusTab()` with the `tabId` that's returned by the `openTab()` method.

```
import { LightningElement } from 'lwc';
import { openTab, focusTab } from 'lightning/platformWorkspaceApi';

export default class MyComponent extends LightningElement {
    focusNewTab(event) {
        openTab({
            url: '/lightning/r/Account/001R0000003HgssIAC/view',
            label: 'Global Media'
        }).then((tabId) => {
            focusTab(tabId);
        }).catch((error) => {
            console.log(error);
        });
    }
}
```

You can also simplify the JavaScript promise as follows.

```
openTab({
    url: '/lightning/r/Account/001R0000003HgssIAC/view',
    label: 'Global Media',
    focus: true
}).catch((error) => {
    console.log(error);
});
```

Use JavaScript Promises in Aura

Here's the same example as using JavaScript promises in LWC, written for Aura components.

```
({
    focusNewTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            url: '#/sObject/001R0000003HgssIAC/view',
            label: 'Global Media'
        }).then(function(response) {
            workspaceAPI.focusTab({tabId : response});
       })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Error Handling with Promises

Promises can simplify code that handles the success or failure of asynchronous calls. To use error handling with promises, use the `catch()` method on the promise that is returned from calling an API method.

The `catch()` method returns a promise and accepts a single function parameter that's called if the promise is rejected. This function has one argument that shows the reason for the rejection. The promise returned by `catch()` is rejected if the function that is passed in either throws an error or returns a promise that's rejected. Otherwise, the promise is resolved.

Using Background Utility Items

Implement the `lightning:backgroundUtilityItem` interface to create a component that fires and responds to events without rendering in the utility bar.

Note: Lightning Web Components (LWC) doesn't currently support working with background utility items.

This component implements `lightning:backgroundUtilityItem` and listens for `lightning:tabCreated` events when the app loads. The component prevents more than 5 tabs from opening.

```
<aura:component implements="lightning:backgroundUtilityItem">
    <aura:attribute name="limit" default="5" type="Integer" />
    <aura:handler event="lightning:tabCreated" action="{!c.onTabCreated}" />
    <lightning:workspaceAPI aura:id="workspace" />
</aura:component>
```

When a tab is created, the event handler calls `onTabCreated` in the component's controller and checks how many tabs are open. If the number of tabs is more than 5, the leftmost tab automatically closes.

```
({
    onTabCreated: function(cmp) {
        var workspace = cmp.find("workspace");
        var limit = cmp.get("v.limit");
        workspace.getAllTabInfo().then(function (tabInfo) {
            if (tabInfo.length > limit) {
                workspace.closeTab({
```

```
                    tabId: tabInfo[0].tabId
                });
            }
        });
    }
})
```

Background utility items are added to an app the same way normal utility items are, but they don't appear in the utility bar. The icon appears next to background utility items on the utility item list. If you have only background utility items in your utility bar, the utility bar doesn't appear in your app. You need at least one non-background utility item in your utility bar for it to appear.

SEE ALSO:

Salesforce Help: Add a Utility Bar to Lightning Apps

Using Pop-Out Utilities

Utilities that support pop-out can be "popped out" of the utility bar and into their own separate child windows.

To pop a utility out, click the icon. From there, you can pop the utility back into the utility bar with the icon, or close the utility. Pop-out utilities are the Lightning equivalent to multi-monitor components in Classic.

Note: Popping-out docked utility bar items isn't supported in Lightning Experience on iPad Safari.

Standard Utilities

Pop-out is supported for these standard utilities. Standard utilities are utilities that are included with Salesforce.

- Open CTI Softphone
- History
- Rich Text
- Report Chart
- Visualforce
- Flow
- List View
- Recent Items
- Chatter Feed
- Chatter Publisher
- Notes

Custom Utilities

Pop-out is available for custom utilities. To enable pop-out for custom utilities, activate the **Utility Bar: Enable Pop-Out for Custom Utilities** critical update. The critical update enables pop-out for all utilities in the "Custom" and "Custom – Managed" categories. Test your custom utilities in a sandbox environment before you enable the update.

Disabling Pop-Out

If you don't want your custom utility to be popped out, you can disable pop-out in two ways.

Disabling Pop-Out within the Component

Use the `lightning:utilityItem` interface in your component and set the `supportsPopOut` attribute to `false` to disable pop-out.

```
<aura:component implements="lightning:utilityItem">
    <aura:attribute name="supportsPopOut" type="Boolean" default="false" />
</aura:component>
```

Disabling pop-out within the component itself is a useful and simple way to ensure that the component can never be popped out.

Disabling Pop-Out with the Lightning Console JavaScript API

Use the `disableUtilityPopOut()` method and set the `disabled` argument to `true` to disable utility pop-out.

If you're migrating from a Classic console app and using a Visualforce page for your utility, we automatically respect if `setCustomConsoleComponentPopoutable` is set to false.

Disabling pop-out with the Lightning Console JavaScript API allows you to enable and disable pop-out in real time.

IN THIS SECTION:

Supported APIs
A list of methods and events that support utility pop-out.

SEE ALSO:

disableUtilityPopOut() for Lightning Experience

Supported APIs

A list of methods and events that support utility pop-out.

Note: Custom events aren't supported while a utility is popped out. If custom events are critical to your utility's functionality, we recommend disabling pop-out for your utility.

Lightning Web Components (LWC) doesn't currently support working with background utility items.

Lightning Console JavaScript API Methods for Navigation Items

Methods	Supports Pop-Out	Notes
focusNavigationItem() for Lightning Experience	✔	
getNavigationItems() for Lightning Experience	✔	
getSelectedNavigationItem() for Lightning Experience	✔	
refreshNavigationItem() for Lightning Experience	✔	
setSelectedNavigationItem() for Lightning Experience	✔	

Lightning Console JavaScript API Methods for Workspace Tabs and Subtabs

Methods	Supports Pop-Out	Notes
closeTab() for Lightning Experience	✔	
disableTabClose() for Lightning Experience	✔	
focusTab() for Lightning Experience	✔	
generateConsoleUrl() for Lightning Experience	✔	
getAllTabInfo() for Lightning Experience	✔	
getEnclosingTabId() for Lightning Experience	✔	
getFocusedTabInfo() for Lightning Experience	✔	
getTabInfo() for Lightning Experience	✔	
getTabURL() for Lightning Experience	✔	
isConsoleNavigation() for Lightning Experience	✔	
isSubtab() for Lightning Experience	✔	
openConsoleUrl() for Lightning Experience	✔	
openSubtab() for Lightning Experience	✔	
openTab() for Lightning Experience	✔	
refreshTab() for Lightning Experience	✔	
setTabHighlighted() for Lightning Experience	✔	
setTabIcon() for Lightning Experience	✔	
setTabLabel() for Lightning Experience	✔	

Lightning Console JavaScript API Methods for the Utility Bar

Methods	Supports Pop-Out	Notes
getAllUtilityInfo() for Lightning Experience	✔	
getEnclosingUtilityId() for Lightning Experience	✔	
getUtilityInfo() for Lightning Experience	✔	
minimizeUtility() for Lightning Experience	✔	Returns `false` when popped out

Methods	Supports Pop-Out	Notes
onUtilityClick() for Lightning Experience	✗	
openUtility() for Lightning Experience	✓	
setPanelHeaderIcon() for Lightning Experience	✓	
setPanelHeaderLabel() for Lightning Experience	✓	
setPanelHeight() for Lightning Experience	✓	
setPanelWidth() for Lightning Experience	✓	
setUtilityHighlighted() for Lightning Experience	✓	
setUtilityIcon() for Lightning Experience	✓	
setUtilityLabel() for Lightning Experience	✓	
toggleModalMode() for Lightning Experience	✗	

Lightning Console JavaScript API Events

Events	Supports Pop-Out	Notes
lightning:tabClosed	✓	
lightning:tabCreated	✓	
lightning:tabFocused	✓	
lightning:tabRefreshed	✓	
lightning:tabReplaced	✓	
lightning:tabUpdated	✓	

Salesforce Classic Console API Methods for Primary Tabs and Subtabs

Methods	Supports Pop-Out	Notes
isInConsole()	✓	

Salesforce Classic Console API Events

Events	Supports Pop-Out	Notes
removeEventListener()	✓	

Events	Supports Pop-Out	Notes
fireEvent()	✔	`fireEvent()` returns `success true` even when eventListeners for the given eventType are removed.
addEventListener()	✔	

Using Events with the Lightning Console JavaScript API

The Lightning framework uses event-driven programming, which allows you to create handlers to respond to interface events as they occur. The Lightning Console JavaScript API provides several events specific to Lightning console apps.

Work with Events in Lightning Web Components (LWC)

A Lightning Message Service (LMS) channel is created for each of Aura tab events on page 136. The payloads on the LMS channels are the same as those on the Aura events. Subscribe to the Lightning message channels corresponding to the Aura application events you want to listen for.

Table 1: Aura Events and LMS Channels

Aura Event	LMS Channel	Payload
lightning:tabClosed	`lightning__tabClosed`	`tabId`
lightning:tabCreated	`lightning__tabCreated`	`tabId`
lightning:tabFocused	`lightning__tabFocused`	`previousTabId, currentTabId`
lightning:tabRefreshed	`lightning__tabRefreshed`	`tabId`
lightning:tabReplaced	`lightning__tabReplaced`	`tabId`
lightning:tabUpdated	`lightning__tabUpdated`	`tabId`

Subscribe to LMS Channels in LWC

To subscribe to an LMS channel, import the `lightning/messageService` module and the channel you want. This example imports the `@salesforce/messageChannel/lightning__tabCreated` channel and subscribes to messages that are published over the channel.

Subscribe when the component is created and unsubscribe when the component is destroyed. For more information, see Subscribe and Unsubscribe from a Message Channel.

```
import { LightningElement, wire} from 'lwc';
import { MessageContext, subscribe, unsubscribe } from 'lightning/messageService';
import tabCreatedChannel from "@salesforce/messageChannel/lightning__tabCreated";

export default class MyComponent extends LightningElement {
   @wire(MessageContext) messageContext;
   messageSubscription = null;

   connectedCallback() {
```

```
      this.unsubscribe();
     this.messageSubscription = subscribe(this.messageContext, tabCreatedChannel, (message)
 => {
        this.handleMessage(message);
      });
   }
   disconnectedCallback() {
      this.unsubscribe();
   }

   unsubscribe() {
      if (!this.messageSubscription) {
         return;
      }
      unsubscribe(this.messageSubscription);
      this.messageSubscriptions = null;
   }

   handleMessage(message) {
     if (!message || !message.tabId) {
        return;
     }
     const tabId = { message };
     console.log(`Tab with tabId of ${tabId} is created.`);
   }
}
```

Work with Events in Aura Components

Events are fired from JavaScript controller actions. Events can contain attributes that can be set before the event is fired and read when the event is handled. Each event that works with Lightning console apps returns attributes that can be read once the event is fired. See the reference section of this guide for a list of attributes returned by each event.

To use console events, set up a handler in your Aura component. The following handler, for example, listens for the `lightning:tabCreated` event, and calls the `onTabCreated` function in the component's controller when the event occurs.

```
<aura:handler event="lightning:tabCreated" action="{! c.onTabCreated }"/>
```

Let's look at a more fleshed out example. The following component uses the `lightning:tabClosed` event.

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
<aura:handler event="lightning:tabClosed" action="{! c.onTabClosed }"/>
</aura:component>
```

When a tab is closed, the event handler calls `onTabClosed` in the component's controller, which logs the `tabId` of the closed tab.

```
({
  onTabClosed : function(component, event, helper) {
    var tabId = event.getParam("tabId");
    alert("Tab with tabId of " + tabId + " was just closed.");
  }
})
```

You can use Lightning console events with the Workspace API and Utility Bar API to customize your users' experience. You can, for example, give a tab focus when it's refreshed, or notify the user with a modal dialogue when a tab is replaced.

SEE ALSO:

- Events for Lightning Console JavaScript API
- Trailhead: Connect Components with Events
- *Lightning Aura Components Developer Guide*: Communicating with Events

Using Page Context in the Utility Bar API

In both Lightning console apps and standard navigation apps, utilities can respond to the context of the current page. For a Lightning web component, use the `CurrentPageReference` wire adapter. For an Aura component, specify `implements="force:hasRecordId"` to access the `recordId` of the record a user is viewing.

Lightning Web Components (LWC)

The `CurrentPageReference` wire adapter provides the page reference that describes the current page.

```
import { LightningElement, wire } from 'lwc';
import { CurrentPageReference } from "lightning/navigation";

export default class UtilityBarRecordExample extends LightningElement {
    recordId;
    isRecord = false;

    @wire(CurrentPageReference)
    wireCurrentPageReference(currentPageReference) {
        if (currentPageReference) {
            this.recordId = currentPageReference.attributes.recordId;
            this.isRecord = true;
        }
    }
}
```

The component displays the record ID when the utility bar is loaded on a record page.

```
<template>
    <div class="slds-m-around_medium">
        <p lwc:if={isRecord}>You are viewing record: {recordId}</p>
    </div>
</template>
```

For more information about working with the page reference, see the LWC Dev Guide.

Aura Components

This simple component implements `force:hasRecordId` and listens for changes to the record being viewed. When this component is added to a utility bar, it displays the `recordId` of the record currently being viewed.

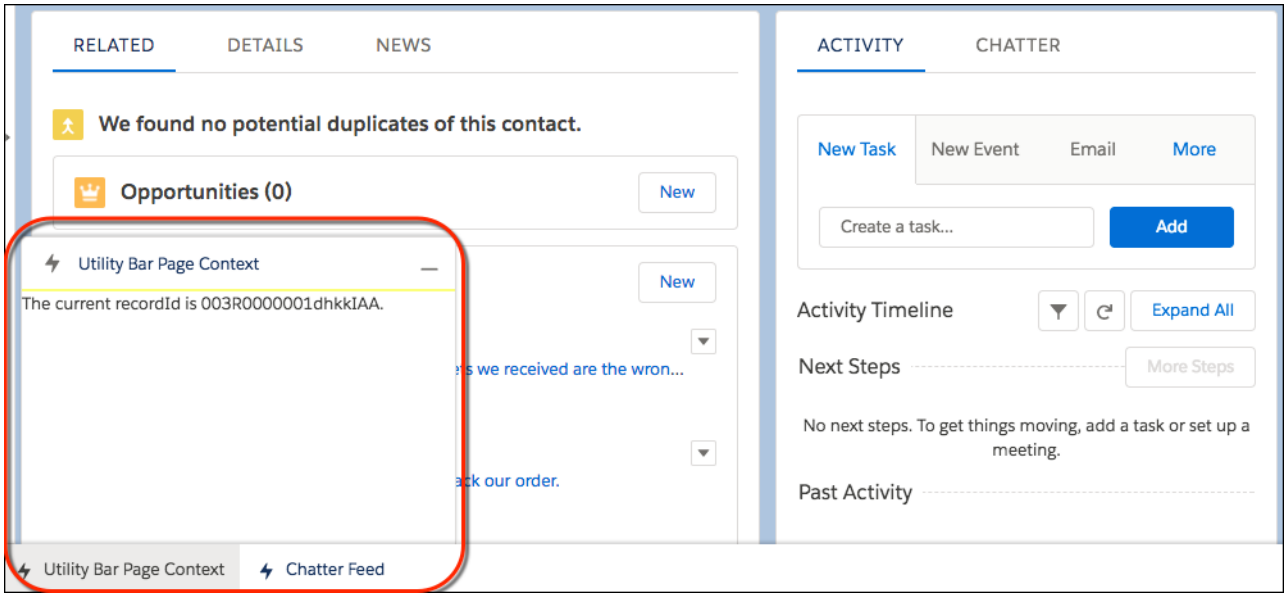
```
<aura:component implements="force:hasRecordId,flexipage:availableForAllPageTypes"
access="global">
    <aura:handler name="change" value="{!v.recordId}" action="{!c.onRecordIdChange}"/>
```

```
    <div>
            <p>The current recordId is {!v.recordId}.</p>
    </div>
</aura:component>
```

The component's controller listens for changes to the `recordId`, and prints the new `recordId` to the developer console upon a change.

```
({
    onRecordIdChange : function(component, event, helper) {
        var newRecordId = component.get("v.recordId");
        console.log(newRecordId);
    }
})
```

This image shows what the component looks like in the utility bar of a Lightning console app.



Using Page References to Open Console Workspace Tabs and Subtabs

You can navigate to different page types, including a URL addressable custom component. To make a custom component URL addressable using LWC, use the `lightning__UrlAddressable` target. To make an Aura component URL addressable, implement the `lightning:isUrlAddressable` interface on your custom component.

Use Page References in LWC

You can use a page reference to open different page types in a tab or subtab. This example opens a subtab using openTab() to display the view page on a specified PersonAccount record.

```
import { LightningElement, wire } from 'lwc';
import { EnclosingTabId, openSubtab } from 'lightning/platformWorkspaceApi';
```

```
export default class MyComponent extends LightningElement {
   @wire(EnclosingTabId) enclosingTabId;

   openAnotherSubTab() {
     if (!this.enclosingTabId) {
        return;
     }
     openSubtab(this.enclosingTabId, {
         pageReference: {
            type: 'standard__objectPage',
            attributes: {
                recordId: '001xx000003DGg0AAG',
                objectApiName: 'PersonAccount',
                actionName: 'view'
            }
         }
      });
   }
}
```

Navigate to a URL Addressable Component in LWC

Making a component URL addressable provides the following benefits for console apps:

- Future-proofs your apps from changes in URL formats.
- Generates a user-friendly URL for your tabs.
- Opens an Aura component as a subtab, even if called from a utility, a hover, or another page.
- Allows a mechanism to conditionally open a given component more than once or redirect to an already open workspace or subtab using the `uid` parameter.

  Warning: Other uses for the `uid` parameter that are not explicitly outlined in this document are not supported.

For example, you have a URL addressable `myComponent` component, and a `workspaceOpenTab` component that navigates to the addressable component.

To make `myComponent` available for navigation, set the `<isExposed>` tag to true in the `myComponent.js-meta.xml` configuration file. The `<apiVersion>` tag has no impact on the `lightning__UrlAddressable` target and can be set to an earlier version.

```
<!-- myComponent.js-meta.xml -->
<?xml version="1.0" encoding="UTF-8"?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>61.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__UrlAddressable</target>
    </targets>
</LightningComponentBundle>
```

`myComponent` displays the URL and page reference information that the `workspaceOpenTab` component passes in.

```
<!-- myComponent.html -->
<template>
  <div class="slds-var-m-around_medium">
```

```
    <p>Component URL: {connectedCallbackUrl}</p>
    <p>Current page reference:</p>
    <pre><code>{currentPageRefFormatted}</code></pre>
  </div>
</template>
```

The component's JavaScript uses the `CurrentPageReference` wire adapter to return page reference information. In this example, the URL returns `https://MyDomainName.my.salesforce.com/lightning/cmp/c__myComponent?c__mystate=value&uid=_uniqueId_`.

```
// myComponent.js
import { LightningElement, wire } from 'lwc';
import { CurrentPageReference } from 'lightning/navigation';

export default class MyComponent extends LightningElement {
  @wire(CurrentPageReference)
  currentPageRef;

  connectedCallbackUrl;

  connectedCallback() {
    this.connectedCallbackUrl = window.location.href;
  }

  get currentPageRefFormatted() {
    return JSON.stringify(this.currentPageRef, undefined, 2);
  }
}
```

This `workspaceOpenTab` has a button that opens the URL addressable component in a new workspace tab. Its `.js-meta.xml` configuration file includes the `lightning__AppPage` target only. It assumes that you add the `workspaceOpenTab` component to a Lightning console app.

```
<!-- workspaceOpenTab.html -->
<template>
  <div class="slds-m-around_medium">
      <lightning-button label="Open Tab" onclick={handleOpen}>
      </lightning-button>
  </div>
</template>
```

The component's JavaScript calls the openTab() method from `lightning/platformWorkspaceApi`. To prevent the app from opening a new tab if the tab with the component is already opened, pass in a `uid` value to the `state` object.

```
// workspaceOpenTab.js
import { LightningElement, wire } from 'lwc';
import { NavigationMixin } from 'lightning/navigation';
import { IsConsoleNavigation, openTab } from 'lightning/platformWorkspaceApi';

export default class WorkspaceOpenTab extends NavigationMixin(LightningElement) {
  @wire(IsConsoleNavigation) isConsoleNavigation;

  async handleOpen() {
    if (!this.isConsoleNavigation) {
      return;
```

```
    }
    try {
      await openTab({
        pageReference: {
          type: 'standard__component',
          attributes: {
            componentName: 'c__myComponent',
          },
          state: {
            c__stateKey: 'stateValue',
            uid: "1",
          },
        },
        icon: 'utility:sparkles',
        label: 'My Component',
      });
    } catch (error) {
      // handle error
    }
  }
}
```

Use Page References in Aura Components

To create a page reference we can use to open workspace tabs and subtabs, let's create `greetings.cmp`, and implement `lightning:isUrlAddressable`. This component displays "Hello, <name>" where a URL parameter, `c__name`, provides the name when the component is opened. The component also defines a `pageReference` that we can use to navigate to it.

```
<aura:component implements="lightning:isUrlAddressable">
    <aura:attribute name="name" type="String" description="The person that will be greeted"
 />
    <aura:handler name="init" value="{!this}" action="{!c.init}" />
    <aura:handler name="change" value="{!v.pageReference}" action="{!c.handlePageChange}"
 />
    <h1>Greeting Page</h1>
    <div>Hello, {!v.name}</div>
</aura:component>
```

The JavaScript controller `greetingsController.js` handles URL parameters in the `init` method and assigns the `name` attribute using that URL parameter.

```
({
    init: function(cmp, evt, hlp) {
        var myPageRef = cmp.get("v.pageReference");
        var name = myPageRef && myPageRef.state ? myPageRef.state.c__name : "World";
        cmp.set("v.name", name);
    },
    handlePageChange: function(cmp, evt, hlp) {
        var myPageRef = cmp.get("v.pageReference");
        var name = myPageRef && myPageRef.state ? myPageRef.state.c__name : "World";
        cmp.set("v.name", name);
    }
})
```

Now let's create `openGreetings.cmp`, which includes an input field to set the `c__name` URL parameter when we open `greetings.cmp`.

```
<aura:component>
    <aura:attribute name="pageReference" type="Object"/>
    <lightning:workspaceAPI aura:id="workspace"/>
    <lightning:button label="Open Greeting in Subtab" onclick="{!c.openSubtab}"/>
    <lightning:input label="Name" name="myname"/>
</aura:component>
```

The controller `openGreetingsController.js` uses `openSubtab()` on page 63 and sets `c__name` to the value of the `myname` input field. You can use the `uid` parameter to conditionally dedupe tabs and subtabs. Omit the `uid` to open a new tab or subtab every time.

```
({
    openSubtab: function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getEnclosingTabId().then(function(enclosingTabId) {
            workspaceAPI.openSubtab({
                parentTabId: enclosingTabId,
                pageReference: {
                    "type": "standard__component",
                    "attributes": {
                        "componentName": "c__greetings"
                    },
                    "state": {
                        "uid": "1",
                        "c__name": component.get("v.myName")
                    }
                }
            }).then(function(subtabId) {
                console.log("The new subtab ID is:" + subtabId);
            }).catch(function(error) {
                console.log("error");
            });
        });
    }
})
```

Now that we have everything set up, we can test our components by creating a custom tab in Setup for `openGreetings.cmp`. Add the custom tab to a console app and open the console app. Select the custom tab from the nav menu to open `openGreetings.cmp`. Enter a name and click "Open Greeting in Subtab." `greetings.cmp` opens as a subtab and displays its greeting with the provided name.

SEE ALSO:

Lightning Web Components Developer Guide: Basic Navigation

Lightning Web Components Developer Guide: `pageReference` Types

Aura Components Developer Guide: Navigate Across Your Apps with Page References

Debugging

Use your browser's console and JavaScript error messages generated within Salesforce to debug Lightning pages built with the Lightning Console JavaScript API. The methods in the Lightning Console JavaScript APIs are asynchronous and return their results using promises.

To print messages to your browser's console, use `console.log()` in your component controller code.

Salesforce also displays JavaScript errors at runtime, which provide the stack trace when there's a bug.

Methods for Lightning Console JavaScript API

If your org is using Lightning Experience, use Lightning Console JavaScript API methods.

IN THIS SECTION:

Methods for Navigation Items in Lightning Experience

Lightning console apps display an item menu that lets users select navigation items, such as cases, contacts, and accounts. Salesforce admins choose which navigation items to display in the navigation menu.

These methods work with navigation items in Lightning console apps.

IN THIS SECTION:

`focusNavigationItem()` for Lightning Experience

Focuses on the selected navigation object and opens the object's home page. Typically, standard and custom objects open the object's list view. If split view is open, focus remains on the selected navigation object. This method works only in Lightning console apps.

Keep these things in mind when working with this method.

- If a tab is already open for the navigation item, the focus is set on the tab.
- If split view is open, the focus is set on the navigation tab.
- If split view is collapsed, the navigation item's tab is opened and focus is set on the tab.

Arguments

None

Sample Code

This component has a button that, when pressed, focuses on the navigation item and opens the navigation item's home page. For most objects, the home page is the object's list view.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:navigationItemAPI aura:id="navigationItemAPI"/>
    <lightning:button label="Focus navigation item" onclick="{!c.focusNavigationItem}"/>
</aura:component>
```

Controller code:

```
({
    focusNavigationItem : function(component, event, helper) {
        var navigationItemAPI = component.find("navigationItemAPI");
        navigationItemAPI.focusNavigationItem().then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
```

```
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`. The promise is rejected on error.

`getNavigationItems()` for Lightning Experience

Returns information about all the items in the navigation menu. This method works only in Lightning console apps.

Arguments

None

Sample Code

This component has a button that, when pressed, returns information about the navigation items in a console app.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:navigationItemAPI aura:id="navigationItemAPI"/>
    <lightning:button label="Get navigation item" onclick="{!c.getNavigationItems}"/>
</aura:component>
```

Controller code:

```
({
    getNavigationItems : function(component, event, helper) {
        var navigationItemAPI = component.find("navigationItemAPI");
        navigationItemAPI.getNavigationItems().then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to an array of `navigationItemInfo` objects. The promise is rejected on error.

The `navigationItemInfo` object contains the following fields.

Name	Type	Description
`label`	string	The navigation item's label, such as `Account` or `Case`.

Name	Type	Description
`developerName`	string	The navigation item's developer name that uniquely identifies the item. For example, `Salesforce_Account` or `Your_VF_Page_Name`.
`selected`	boolean	True if the tab is currently selected, false otherwise.
`pageReference`	object	The representation of the current page. The object returns information such as: page type (for example `standard__objectPage` or `standard__navItemPage`), object API name, and state information for the page.

Here's the structure of a `navigationItemInfo` object.

```
{
      developerName : string,
      label : string,
      pageReference: object,
      selected : boolean
}
```

`getSelectedNavigationItem()` for Lightning Experience

Returns information about the selected navigation item. This method works only in Lightning console apps.

Arguments

None

Sample Code

This component has a button that, when pressed, returns information about the selected navigation item.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:navigationItemAPI aura:id="navigationItemAPI"/>
    <lightning:button label="Get selected navigation item"
onclick="{!c.getSelectedNavigationItem}"/>
</aura:component>
```

Controller code:

```
({
    getSelectedNavigationItem : function(component, event, helper) {
        var navigationItemAPI = component.find("navigationItemAPI");
        navigationItemAPI.getSelectedNavigationItem().then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
            console.log(error);
```

```
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to a `navigationItemInfo` object. The promise is rejected on error.

The `navigationItemInfo` object has the following fields.

Name	Type	Description
`label`	string	The navigation item's label, such as `Account` or `Case`.
`developerName`	string	The navigation item's developer name that uniquely identifies the item. For example, `Salesforce_Account` or `Your_VF_Page_Name`.
`selected`	boolean	True if the tab is currently selected, false otherwise.
`pageReference`	object	The representation of the current page. The object returns information such as: page type (for example `standard__objectPage` or `standard__navItemPage`), object API name, and state information for the page.

Here's the structure of a `navigationItemInfo` object.

```
{
      developerName : string,
      label : string,
      pageReference: object,
      selected : boolean
}
```

`refreshNavigationItem()` for Lightning Experience

Refreshes the selected navigation object's home page. Typically, standard and custom objects open the object's list view. If split view is open, it's refreshed. This method works only in Lightning console apps.

This method refreshes in the background. If the list view has unsaved changes, the method returns false and doesn't refresh the navigation item. The method doesn't set focus on the navigation tab.

The following navigation items aren't supported:

- Custom Visualforce tabs
- Custom Aura component tabs
- Custom web tabs
- Dashboards
- Reports

Arguments

None

Sample Code

This Aura component has a button that, when pressed, refreshes the navigation item.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:navigationItemAPI aura:id="navigationItemAPI"/>
   <lightning:button label="Refresh navigation item" onclick="{!c.refreshNavigationItem}"/>
</aura:component>
```

Controller code:

```
({
    refreshNavigationItem : function(component, event, helper) {
        var navigationItemAPI = component.find("navigationItemAPI");
        navigationItemAPI.refreshNavigationItem().then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`. The promise is rejected on error.

`setSelectedNavigationItem()` for Lightning Experience

Sets the selected navigation item to a specific ID. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`developerName`	string	The ID of the navigation item.

Sample Code

This Aura component has a button that, when pressed, sets the specified ID as the selected navigation item.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" description="My Lightning
 Component">
    <lightning:navigationItemAPI aura:id="navigationItemAPI" />
    <lightning:button label="Set Navigation Item" onclick="{! c.setSelectedNavigationItem
```

```
 }" />
</aura:component>
```

Controller code:

```
({
    setSelectedNavigationItem : function(component, event, helper) {
        var navigationItemAPI = component.find("navigationItemAPI");
        navigationItemAPI.setSelectedNavigationItem({
            "developerName": "standard-Account"
        }).then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to true.

Methods for Workspace Tabs and Subtabs in Lightning Experience

A Lightning console app displays Salesforce pages as workspace tabs or subtabs. A workspace tab displays the main work item or record, such as an account. A subtab displays related records, such as an account's contacts or opportunities.

These methods work with workspace tabs and subtabs in Lightning console apps. Both Lightning Web Components (LWC) and Aura Components are supported unless otherwise specified.

Note: Keep in mind that tabIds are case sensitive.

IN THIS SECTION:

addToBrowserTitleQueue() for Lightning Experience

Adds a string to a list of titles that rotate in the browser title bar every three seconds.This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

closeTab() for Lightning Experience

Closes a workspace tab or subtab. This method works only in Lightning console apps.

disableTabClose() for Lightning Experience

Prevents a workspace tab or subtab from closing. This method removes the close button from a tab or subtab, and disables the keyboard shortcuts that close tabs and subtabs. This method works only in Lightning console apps.

focusTab() for Lightning Experience

Focuses a workspace tab or subtab. This method works only in Lightning console apps.

generateConsoleUrl() for Lightning Experience

Generates a URL for a workspace tab and its subtabs. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

getAllTabInfo() for Lightning Experience

Returns information about all open tabs. This method works only in Lightning console apps.

getEnclosingTabId() for Lightning Experience

Returns the ID of the enclosing tab. This method isn't supported for Lightning Web Components (LWC).

getFocusedTabInfo() for Lightning Experience

Returns information about the focused workspace tab or subtab. This method works only in Lightning console apps.

getTabInfo() for Lightning Experience

Returns information about the specified tab. This method works only in Lightning console apps.

getTabURL() for Lightning Experience

Returns the URL of the specified tab. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

isConsoleNavigation() for Lightning Experience

Determines whether the app it's used within uses console navigation. This method isn't supported for Lightning Web Components (LWC).

isSubtab() for Lightning Experience

Checks whether a tab is a subtab. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

openConsoleUrl() for Lightning Experience

Opens a URL generated by `generateConsoleUrl()`. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

openSubtab() for Lightning Experience

Opens a subtab within a workspace tab. If the subtab is already open, the subtab is focused. This method works only in Lightning console apps.

openTab() for Lightning Experience

Opens a new workspace tab. If the tab is already open, the tab is focused.

refreshTab() for Lightning Experience

Refreshes a workspace tab or a subtab specified by `tabId`. Keep in mind that the first subtab has the same `tabId` as the workspace tab. This method works only in Lightning console apps.

removeFromBrowserTitleQueue() for Lightning Experience

Removes a string from a list of titles that rotate in the browser title bar every three seconds. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

setTabHighlighted() for Lightning Experience

Highlights the specified tab with a different background color and a badge. Tab highlights don't persist after reloading a Lightning console app. This method works only in Lightning console apps.

setTabIcon() for Lightning Experience

Sets the icon and alternative text of the specified tab. This method works only in Lightning console apps.

setTabLabel() for Lightning Experience

Sets the label of the specified tab. This method works only in Lightning console apps.

EnclosingTabId Context Wire Adapter for Lightning Experience

Returns the ID of the enclosing tab or subtab. This wire adapter is available for Lightning Web Components (LWC) only.

IsConsoleNavigation Context Wire Adapter for Lightning Experience

Determines whether the app it's used within uses console navigation. This wire adapter is available for Lightning Web Components (LWC) only.

`addToBrowserTitleQueue()` for Lightning Experience

Adds a string to a list of titles that rotate in the browser title bar every three seconds.This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

Note: Accurate browser tab titles help improve accessibility. Screen readers announce page titles when a page is first loaded, and don't announce dynamic updates to the title. Use the root node of the document, like `document.title`, to announce the updated browser tab title instead.

Arguments

Name	Type	Description
`title`	string	The browser tab title to add.

Aura Components Sample Code

This component has a button that, when pressed, adds a string to a list of titles that rotate in the browser title bar every three seconds.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Add to Browser Title Queue" onclick="{!
c.handleAddToBrowserTitleQueue }" />
</aura:component>
```

Controller code:

```
({
    handleAddToBrowserTitleQueue : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.addToBrowserTitleQueue({
            title: "New Browser Title"
        })
        .then(function(result){
            console.log(result);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`closeTab()` for Lightning Experience

Closes a workspace tab or subtab. This method works only in Lightning console apps.

Arguments

The method provides the same argument for both Aura Components and Lightning Web Components (LWC).

Name	Type	Description
`tabId`	string	ID of the workspace tab or subtab to close.

LWC Sample Code

This component checks if it's in a Lightning console app using the `isConsoleNavigation` wire adapter. When the `getFocusedTabInfo()` on page 54 resolves successfully, it returns the `tabInfo` object. The `const { tabId }` syntax destructures the `tabInfo` object and binds the `tabInfo.tabId` value on the `tabId` variable. `closeTab()` uses this `tabId` value to close the tab.

```
import { LightningElement, wire } from 'lwc';
import { IsConsoleNavigation, getFocusedTabInfo, closeTab } from
'lightning/platformWorkspaceApi';

export class WorkspaceAPICloseTab extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async closeTab() {
        if (!this.isConsoleNavigation) {
            return;
        }
        const { tabId } = await getFocusedTabInfo();
        await closeTab(tabId);
    }
}
```

This example shows the workspaceAPICloseTab component in the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, closes the currently focused tab.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:workspaceAPI aura:id="workspace"/>
    <lightning:button label="Close Focused Tab" onclick="{!c.closeFocusedTab}"/>
</aura:component>
```

Controller code:

```
({
    closeFocusedTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
            var focusedTabId = response.tabId;
            workspaceAPI.closeTab({tabId: focusedTabId});
        })
```

```
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that resolves to `true` if successful. The promise is rejected on error.

`disableTabClose()` for Lightning Experience

Prevents a workspace tab or subtab from closing. This method removes the close button from a tab or subtab, and disables the keyboard shortcuts that close tabs and subtabs. This method works only in Lightning console apps.

Arguments

The method provides the same argument for both Aura Components and Lightning Web Components (LWC).

Name	Type	Description
`tabId`	string	The ID of the workspace tab or subtab to disable tab close for.
`disabled`	boolean	Specifies whether to disable tab close.

Note: `disableTabClose()` doesn't prevent the browser from refreshing or closing the browser tab.

LWC Sample Code

This component has a function that opens a tab using a page reference and then prevents it from being closed. When the `openTab()` method resolves successfully, it returns the `tabInfo` object. The `const { tabId }` syntax destructures the `tabInfo` object and binds the `tabInfo.tabId` value on the `tabId` variable. `disableTabClose()` uses this `tabId` value to prevent the tab from closing.

```
import { LightningElement } from 'lwc';
import { openTab, disableTabClose, IsConsoleNavigation } from
'lightning/platformWorkspaceApi';

export default class DisableTabCloseExample extends LightningElement {

   async handleOpenAndDisable() {
        if (!this.isConsoleNavigation) {
            return;
        }
        try {
            const { tabId } = await openTab({
                pageReference: {
                    "type": "standard__objectPage",
                    "attributes": {
                        "objectApiName":"Account",
```

```
                        "actionName":"home"
                    },
                },
        });
        await disableTabClose(tabId, true);
        } catch (error) {
            // handle error
        }
    }
}
```

For another example that uses `disableTabClose()`, see the workspaceAPIDisableTabClose component in the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, disables the ability to close the currently focused tab.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:workspaceAPI aura:id="workspace" />
   <lightning:button label="Disable Close Focused Tab" onclick="{! c.disableCloseFocusedTab
 }" />
</aura:component>
```

Controller code:

```
({
    disableCloseFocusedTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
            var focusedTabId = response.tabId;
            workspaceAPI.disableTabClose({
                tabId: focusedTabId,
                disabled: true
        })
        .then(function(tabInfo) {
            console.log(tabInfo);
        })
        .catch(function(error) {
            console.log(error);
        });
    })
    .catch(function(error) {
        console.log(error);
    });
    }
})
```

Response

This method returns a promise that, upon success, resolves to a `tabInfo` object representing the focused tab. A `tabInfo` object is a JSON array of information about a workspace tab, with nested arrays of information on each subtab. This is the structure of a `tabInfo` object.

```
{
     tabId: string,
     url: string (URL),
     pinned: boolean,
     closeable: boolean,
     title: string,
     icon: string (SLDS iconKey),
     iconAlt: string,
     customTitle: string (optional),
     customIcon: string (optional),
     customIconAlt: string (optional),
     highlighted: boolean,
     pageReference: object,
     isSubtab: boolean,
     parentTabId: string,
     subtabs: [
         {
             tabId: string,
             url: string (URL),
             pinned: boolean,
             closeable: boolean,
             title: string,
             icon: string (SLDS iconKey),
             iconAlt: string,
             customTitle: string (optional),
             customIcon: string (optional),
             customIconAlt: string (optional),
             highlighted: boolean,
             pageReference: object,
             isSubtab: boolean,
             parentTabId: string,
             focused: boolean,
             recordId: string,
          },
           ...
     ],
     focused: boolean,
     recordId: string
}
```

`focusTab()` for Lightning Experience

Focuses a workspace tab or subtab. This method works only in Lightning console apps.

Arguments

The method provides the same argument for both Aura Components and Lightning Web Components (LWC).

Name	Type	Description
`tabId`	string	ID of the workspace tab or subtab on which to focus.

LWC Sample Code

This component has a function that retrieves information of all tabs using `getAllTabInfo()` on page 51, which returns an array of `tabInfo` objects. Then, it uses `focusTab(allTabs[0].tabId)` to focus on the first tab in the array.

```
import { LightningElement } from 'lwc';
import { IsConsoleNavigation, getAllTabInfo, focusTab } from
'lightning/platformWorkspaceApi';

export default class FocusTabExample extends LightningElement {

    async handleOpen() {
        if (!this.isConsoleNavigation) {
            return;
        }
        try {
            const tabInfo = await getAllTabInfo();
            await focusTab(tabInfo[0].tabId);
        } catch (error) {
            console.log(error);
        }
    }
}
```

For another example that uses `focusTab()`, see the workspaceAPIFocusTab component in the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, opens a new tab and focuses it.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Focus New Tab" onclick="{! c.focusNewTab }" />
</aura:component>
```

Controller code:

```
({
    focusNewTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            url: '/lightning/r/Account/001xx000003DI05AAG/view',
        }).then(function(response) {
            workspaceAPI.focusTab({tabId : response});
       })
```

```
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Note: The relative URL used in this example is a placeholder. To try this example yourself, use a relative URL with a record ID from your org.

Response

This method returns a promise that, upon success, resolves to `true`.

`generateConsoleUrl()` for Lightning Experience

Generates a URL for a workspace tab and its subtabs. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

Arguments

Name	Type	Description
`pageReferences`	pageReference[]	An array of page references. The first page reference is the workspace tab. Any following page references are subtabs. The last page reference is the focused subtab.

Sample Code

This component has a button that, when pressed, uses the `generateConsoleUrl()` method to create a URL for the provided page references.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspaceAPI" />
    <lightning:button label="Get Console URL" onclick="{! c.handleGenerateConsoleUrl }"
/>
</aura:component>
```

Controller code:

```
({
    handleGenerateConsoleUrl : function(component, event, helper) {
        var workspaceAPI = cmp.find("workspaceAPI");
        workspaceAPI.generateConsoleURL({
            "pageReferences": [
                {
                    "type": "standard__recordPage",
                    "attributes": {
                        "objectApiName": "Account",
                        "actionName": "view",
                        "recordId": "001xx000003DGQXAA4"
```

```
                    },
                    "state": {}
                },
                {
                    "type": "standard__recordPage",
                    "attributes": {
                        "objectApiName": "Account",
                        "actionName": "view",
                        "recordId": "001xx000003DGQWAA4"
                    },
                    "state": {}
                },
                {
                    "type": "standard__recordPage",
                    "attributes": {
                        "objectApiName": "Account",
                        "actionName": "view",
                        "recordId": "001xx000003DGQYAA4"
                    },
                    "state": {}
                }
            ]
        }).then(function(url) {
            console.log(url);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves with the generated URL.

Name	Type	Description
`url`	string	A console URL that represents the array of URLs passed into Salesforce.

`getAllTabInfo()` for Lightning Experience

Returns information about all open tabs. This method works only in Lightning console apps.

Arguments

None.

LWC Sample Code

This component has a function that returns the information on all tabs.

```
import { LightningElement, wire } from 'lwc';
import { IsConsoleNavigation, getAllTabInfo } from 'lightning/platformWorkspaceApi';

export class GetAllTabInfoExample extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async handleOpen() {
        if (!this.isConsoleNavigation) {
            return;
        }
        try {
            const tabInfo = await getAllTabInfo();
            //do something with tabInfo
        } catch (error) {
            console.log(error);
        }
    }
}
```

For another example that uses `getAllTabInfo()`, see focusTab().

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, gets the info of all open tabs and prints the resulting `tabInfo` object.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Get All Tab Info" onclick="{! c.handleGetAllTabInfo }" />
</aura:component>
```

Controller code:

```
({
    handleGetAllTabInfo : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getAllTabInfo().then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to an array of `tabInfo` objects. A `tabInfo` object is a JSON array of information about a workspace tab, with nested arrays of information on each subtab. This is the structure of a `tabInfo` object.

```
{
      tabId: string,
      url: string (URL),
      pinned: boolean,
      closeable: boolean,
      title: string,
      icon: string (SLDS iconKey),
      iconAlt: string,
      customTitle: string (optional),
      customIcon: string (optional),
      customIconAlt: string (optional),
      highlighted: boolean,
      pageReference: object,
      isSubtab: boolean,
      parentTabId: string,
      subtabs: [
          {
              tabId: string,
              url: string (URL),
              pinned: boolean,
              closeable: boolean,
              title: string,
              icon: string (SLDS iconKey),
              iconAlt: string,
              customTitle: string (optional),
              customIcon: string (optional),
              customIconAlt: string (optional),
              highlighted: boolean,
              pageReference: object,
              isSubtab: boolean,
              parentTabId: string,
              focused: boolean,
              recordId: string,
           },
            ...
      ],
      focused: boolean,
      recordId: string
}
```

`getEnclosingTabId()` for Lightning Experience

Returns the ID of the enclosing tab. This method isn't supported for Lightning Web Components (LWC).

To retrieve the enclosing tab ID with LWC, see EnclosingTabId context wire adapter.

Tip: To retrieve information about the tab or the subtab that a component is rendered in, first use `getEnclosingTabId()` instead of `getFocusedTabInfo()` on page 54. Then call `getTabInfo()` on page 57 and use the enclosing tab's ID as the argument. By using `getEnclosingTabId()`, you make sure that the correct tab ID is returned when you work with lifecycle hooks such as `renderedCallback()` or `connectedCallback()`.

Arguments

None.

Aura Components Sample Code

This component has a button that, when clicked, retrieves the enclosing tab ID.

This is the component code. The `lightning:workspaceAPI` component provides access to Lightning console methods. When clicked, the `lightning:button` base component executes the `handleGetEnclosingTabId` action in the component's client-side controller.

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Get Enclosing Tab Id" onclick="{! c.handleGetEnclosingTabId
}" />
</aura:component>
```

This is the controller code. The `handleGetEnclosingTabId` action returns the ID of the enclosing workspace tab.

```
({
    handleGetEnclosingTabId : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getEnclosingTabId().then(function(tabId) {
            console.log(tabId);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to the `tabId` of the enclosing tab, if within a tab. If not within a tab, the method resolves to `false` upon success.

`getFocusedTabInfo()` for Lightning Experience

Returns information about the focused workspace tab or subtab. This method works only in Lightning console apps.

Note: To retrieve information about the workspace tab or the subtab that a component is rendered in, don't use `getFocusedTabInfo()`. When lifecycle hooks such as `renderedCallback()` or `connectedCallback()` are invoked for the component, the tab enclosing that component isn't guaranteed to be in focus, so `getFocusedTabInfo()` sometimes doesn't return that tab's information.

Instead, first use `getEnclosingTabId()` on page 53 for Aura components or the `EnclosingTabId` on page 78 wire adaptor for LWC. Then call `getTabInfo()` on page 57 and use the enclosing tab's ID as the argument.

We recommend that you continue to use `getFocusedTabInfo()` with components in the utility bar, because the utility bar doesn't have an enclosing tab ID.

Arguments

None.

LWC Sample Code

This component checks if it's in a Lightning console app, using the `IsConsoleNavigation` wire adapter, and returns information about the focused tab or subtab. It uses the `tabId` and `highlighted` properties from the `tabInfo` return object to toggle highlighting on the tab.

```
import { LightningElement, wire } from 'lwc';
import { IsConsoleNavigation, getFocusedTabInfo, setTabHighlighted } from
'lightning/platformWorkspaceApi';

export class FocusedTabInfoExample extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async handleFocusToggleHighlight() {
        if (!this.isConsoleNavigation) {
            return;
        }
        try {
            let { tabId, highlighted } = await getFocusedTabInfo();
            highlighted = highlighted ? false : true;
            setTabHighlighted(tabId, highlighted);
        } catch (error) {
            console.log(error);
        }
    }
}
```

For another example, see closeTab().

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when clicked, closes the currently focused tab.

This is the component code. The `lightning:workspaceAPI` component provides access to Lightning console methods. When clicked, the `lightning:button` base component executes the `closeFocusedTab` action in the component's client-side controller.

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Close Focused Tab" onclick="{! c.closeFocusedTab }" />
</aura:component>
```

This is the client-side controller code. The `closeFocusedTab` action retrieves the tab ID of the tab in focus, and then closes the tab with that ID.

```
({
    closeFocusedTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
            var focusedTabId = response.tabId;
            workspaceAPI.closeTab({tabId: focusedTabId});
        })
        .catch(function(error) {
```

```
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to a `tabInfo` object that represents the focused tab. A `tabInfo` object is a JSON array of information about a workspace tab, with nested arrays of information on each subtab. This is the structure of a `tabInfo` object.

```
{
     tabId: string,
     url: string (URL),
     pinned: boolean,
     closeable: boolean,
     title: string,
     icon: string (SLDS iconKey),
     iconAlt: string,
     customTitle: string (optional),
     customIcon: string (optional),
     customIconAlt: string (optional),
     highlighted: boolean,
     pageReference: object,
     isSubtab: boolean,
     parentTabId: string,
     subtabs: [
         {
             tabId: string,
             url: string (URL),
             pinned: boolean,
             closeable: boolean,
             title: string,
             icon: string (SLDS iconKey),
             iconAlt: string,
             customTitle: string (optional),
             customIcon: string (optional),
             customIconAlt: string (optional),
             highlighted: boolean,
             pageReference: object,
             isSubtab: boolean,
             parentTabId: string,
             focused: boolean,
             recordId: string,
          },
           ...
     ],
     focused: boolean,
     recordId: string
}
```

`getTabInfo()` for Lightning Experience

Returns information about the specified tab. This method works only in Lightning console apps.

Arguments

The method provides the same argument for both Aura Components and Lightning Web Components (LWC).

Name	Type	Description
`tabId`	string	ID of the tab for which to retrieve the information.

LWC Sample Code

This component has a function that returns the tab information.

```
import { LightningElement, wire } from 'lwc';
import { EnclosingTabId, getTabInfo } from 'lightning/platformWorkspaceApi';

export default class ConsoleNavExample extends LightningElement {
    @wire(EnclosingTabId) enclosingTabId;
    handleClick() {
        if (this.enclosingTabId) {
            getTabInfo(this.enclosingTabId).then((tabInfo) => {
                // do something with it
            }).catch((error) => {
                console.log(error);
            });
        }
    }
}
```

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, opens a tab and uses the `getTabInfo()` method to print the new tab's `tabInfo` to the developer console.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Get Opened Tab Info" onclick="{! c.getOpenedTabInfo }" />
 </aura:component>
```

Controller code:

```
({
    getOpenedTabInfo : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
```

```
            url: '/lightning/r/Account/001xx000003DI05AAG/view',
            focus: true
        }).then(function(response) {
            workspaceAPI.getTabInfo({
                tabId: response
            }).then(function(response) {
                console.log(response);
            });
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Note: The relative URL used in this example is a placeholder. To try this example yourself, use a relative URL with a record ID from your org.

Response

This method returns a promise that, upon success, resolves to a `tabInfo` object representing the specified tab. A `tabInfo` object is a JSON array of information about a workspace tab, with nested arrays of information on each subtab. This is the structure of a `tabInfo` object.

```
{
     tabId: string,
     url: string (URL),
     pinned: boolean,
     closeable: boolean,
     title: string,
     icon: string (SLDS iconKey),
     iconAlt: string,
     customTitle: string (optional),
     customIcon: string (optional),
     customIconAlt: string (optional),
     highlighted: boolean,
     pageReference: object,
     isSubtab: boolean,
     parentTabId: string,
     subtabs: [
         {
             tabId: string,
             url: string (URL),
             pinned: boolean,
             closeable: boolean,
             title: string,
             icon: string (SLDS iconKey),
             iconAlt: string,
             customTitle: string (optional),
             customIcon: string (optional),
             customIconAlt: string (optional),
             highlighted: boolean,
             pageReference: object,
             isSubtab: boolean,
```

```
            parentTabId: string,
            focused: boolean,
            recordId: string,
        },
         ...
    ],
    focused: boolean,
    recordId: string
}
```

`getTabURL()` for Lightning Experience

Returns the URL of the specified tab. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

To retrieve the URL of a specified tab in LWC, use getTabInfo().

Arguments

Name	Type	Description
`tabId`	string	ID of the tab for which to retrieve the URL.

Aura Components Sample Code

This component has a button that, when pressed, opens a tab and uses the `getTabURL()` method to print the new tab's URL to the developer console.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Get Opened Tab URL" onclick="{! c.getOpenedTabURL }" />
 </aura:component>
```

Controller code:

```
({
    getOpenedTabURL : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            url: '/lightning/r/Account/001xx000003DI05AAG/view',
            focus: true
        }).then(function(response) {
            workspaceAPI.getTabURL({
                tabId: response
            }).then(function(response) {
                console.log(response);
            });
       })
        .catch(function(error) {
            console.log(error);
        });
```

```
    }
})
```

Note: The relative URL used in this example is a placeholder. To try this example yourself, use a relative URL with a record ID from your org.

Response

This method returns a promise that, upon success, resolves to the URL of the specified tab.

`isConsoleNavigation()` for Lightning Experience

Determines whether the app it's used within uses console navigation. This method isn't supported for Lightning Web Components (LWC).

To determine if a component is using console navigation with LWC, see IsConsoleNavigation context wire adapter.

Arguments

None.

Aura Components Sample Code

This component has a button that, when pressed, prints whether the current app is using console navigation.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
   <lightning:button label="Is Console Navigation?" onclick="{! c.handleIsConsoleNavigation
 }" />
</aura:component>
```

Controller code:

```
({
    handleIsConsoleNavigation : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.isConsoleNavigation().then(function(response) {
            console.log(response);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

If the current app uses console navigation, this method returns a promise that resolves to `true` when successful, or `false` otherwise.

`isSubtab()` for Lightning Experience

Checks whether a tab is a subtab. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

To check whether a tab is a subtab in LWC, use getTabInfo().

Arguments

Name	Type	Description
`tabId`	string	ID of the tab.

Aura Components Sample Code

This component has a button that checks whether the focused tab is a subtab and opens a modal with the answer.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
   <lightning:button label="Is the Focused Tab a Subtab?" onclick="{! c.isFocusedTabSubtab
 }" />
 </aura:component>
```

Controller code:

```
({
    isFocusedTabSubtab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
            workspaceAPI.isSubtab({
                tabId: response.tabId
            }).then(function(response) {
                if (response) {
                    confirm("This tab is a subtab.");
                }
                else {
                    confirm("This tab is not a subtab.");
                }
            });
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true` if the tab is a subtab, and `false` otherwise.

`openConsoleUrl()` for Lightning Experience

Opens a URL generated by `generateConsoleUrl()`. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

Arguments

Name	Type	Description
`url`	string	Console URL representing the array of URLs passed into Salesforce.
`focus`	boolean	Optional. If true, the workspace tab opens and displays immediately. If false, the workspace tab opens in the background.
`labels`	string[]	Optional. An array of labels for the opened tabs. The order that the tabs appear in the URL should match the order in the array. Use an emptry string if you don't want to set any labels.

Aura Components Sample Code

This component has a button that, when pressed, opens a workspace using the `openConsoleUrl()` method.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspaceAPI" />
    <lightning:button label="Open Console URL" onclick="{! c.handleOpenConsoleUrl }" />
</aura:component>
```

Controller code:

```
// Assume URL generated by generateConsoleUrl() API
// E.g.
/lightning/r/Account/001x000003DGQAA4/view?ws=%2Flightning%2Fr%2FAccount%2F001x000003DGQAA4%2Fview&tabs=%2Flightning%2Fr%2FAccount%2F001x000003DGQAA4%2Fview&activetab=2
({
    handleOpenConsoleUrl : function(component, event, helper) {
        var url = generateConsoleUrl();
        var workspaceAPI = cmp.find("workspaceAPI");
        workspaceAPI.openConsoleURL({
            "url": url,
            "focus": true,
            "labels": ["Workspace Label", "First Subtab Label", "Second Subtab Label"]
        }).then(function(activeTabId) {
        console.log(activeTabId);
    })
        .catch(function(error) {
            console.log(error);
        })
    }
});
```

Response

This method returns a promise that, upon success, resolves to the `tabId` of the active tab.

`openSubtab()` for Lightning Experience

Opens a subtab within a workspace tab. If the subtab is already open, the subtab is focused. This method works only in Lightning console apps.

Arguments

The method provides the same argument for both Aura Components and Lightning Web Components (LWC). However, `icon`, `iconAlt`, and `label` are supported only for LWC.

Name	Type	Description
`parentTabId`	string	The ID of the workspace tab within which the new subtab opens. You must pass the parent tab ID into the `openSubtab()` method.
`pageReference`	object	Optional. Specifies the `pageReference` to open.
`recordId`	ID	Optional. Specifies the record to open.
`url`	string	Optional. The URL representing the content of the new workspace tab. The URL can be either relative or absolute. To use a third-party domain, add the site as a CSP Trusted Site.
`focus`	boolean	Optional. Specifies whether the new subtab has focus. To display the subtab immediately, set `focus` to `true`. To open the subtab in the background, set `focus` to `false`.
`icon`	string	Optional. An SLDS icon key. See a full list of icon keys on the SLDS reference site. Available for LWC only.
`iconAlt`	string	Optional. Alternative text for the icon. Available for LWC only.
`label`	string	Optional. The text label for the icon. Available for LWC only.

Note: `pageReference`, `recordId`, and `url` are prioritized in that order. Providing arguments with a higher priority means the others get ignored.

LWC Sample Code

This component retrieves the enclosing tab ID using the `EnclosingTabId` wire adapter. It opens a subtab on the current tab when the `handleClick()` function is called. If the component doesn't reside inside a tab or subtab, `tabId` is null.

To check if the current tab is a subtab, use `getTabInfo()`. If the current tab is a subtab, pass in the parent tab ID to the `openSubtab()` function.

```
import { LightningElement, wire } from 'lwc';
import { EnclosingTabId, getTabInfo, openSubtab } from 'lightning/platformWorkspaceApi';

export default class OpenSubtabExample extends LightningElement {
  @wire(EnclosingTabId) tabId;

  async handleClick() {
    if (!this.tabId) {
      return;
    }

    const tabInfo = await getTabInfo(this.tabId);
    const primaryTabId = tabInfo.isSubtab ? tabInfo.parentTabId : tabInfo.tabId;

    // Open a record as a subtab of the current tab
    await openSubtab(primaryTabId, { recordId: '***YourRecordId***', focus: true });
  }
}
```

For another example that uses `openSubtab()`, see the workspaceAPIOpenSubtab component in the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, opens a subtab within a workspace tab.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Open Tab with Subtab" onclick="{! c.openTabWithSubtab }" />
 </aura:component>
```

Controller code:

```
({
    openTabWithSubtab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            url: '/lightning/r/Account/001xx000003DI05AAG/view',
            focus: true
        }).then(function(response) {
            workspaceAPI.openSubtab({
                parentTabId: response,
                url: '/lightning/r/Contact/003xx000004Ts30AAC/view',
                focus: true
            });
```

```
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Note: The relative URL used in this example is a placeholder. To try this example yourself, use a relative URL with a record ID from your org.

Response

This method returns a promise that, upon success, resolves to the ID of the new subtab.

`openTab()` for Lightning Experience

Opens a new workspace tab. If the tab is already open, the tab is focused.

Arguments

The method provides the same argument for both Aura Components and Lightning Web Components (LWC). However, `icon`, `iconAlt`, and `label` are supported only for LWC.

Name	Type	Description
`pageReference`	object	Optional. Generates a unique URL to open. A page reference contains attributes that apply to all pages of that type. See the Aura pageReference types and LWC pageReference types.
`recordId`	ID	Optional. Specifies the record to open.
`url`	URL	Optional. The URL representing the content of the new workspace tab. The URL can be either relative or absolute. To use a third-party domain, add the site as a CSP Trusted Site.
`focus`	boolean	Optional. Specifies whether the new tab has focus. To display the tab immediately, set `focus` to `true`. To open the tab in the background, set `focus` to `false`.
`overrideNavRules`	boolean	Optional. Specifies whether the open tab respects existing navigation rules. This argument has no effect on records that have no navigation rules configured and URLs that do not point to a record.

Name	Type	Description
`icon`	string	Optional. An SLDS icon key in the format `action:canvas` where `action` is the SLDS icon category. See a full list of icon categories and keys on the SLDS reference site. Available for LWC only.
`iconAlt`	string	Optional. Alternative text for the icon. Available for LWC only.
`label`	string	Optional. The text label for the icon. Available for LWC only.

Note: `pageReference`, `recordId`, and `url` are prioritized in that order. Providing an argument with a higher priority means the others are ignored.

LWC Sample Code

This component has a function that opens a specified tab and applies focus on it.

```
import { LightningElement, wire } from 'lwc';
import { IsConsoleNavigation, openTab } from 'lightning/platformWorkspaceApi';

export default class OpenTabExample extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async openTab() {
        if (!this.isConsoleNavigation) {
            return;
        }
        await openTab({
            pageReference: {
                type: 'standard__objectPage',
                attributes: {
                    objectApiName: 'Account',
                    actionName: 'home'
                }
            },
            icon: 'utility:bookmark',
            focus: true,
            label: 'Account List'
        });
    }
}
```

For another example that uses `openTab()`, see the workspaceAPIOpenTab component in the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that when pressed, opens a tab.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
        <lightning:workspaceAPI aura:id="workspace"/>
        <lightning:button label="Open Tab" onclick="{!c.handleOpenTab}"/>
</aura:component>
```

Pass in a pageReference Controller code (pageReference) using the `standard__recordPage` type:

```
({
    handleOpenTab: function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            pageReference: {
                "type": "standard__recordPage",
                "attributes": {
                    "recordId":"500xx000000Ykt2AAC",
                    "actionName":"view"
                },
                "state": {}
            },
            focus: true
        }).then(function(response) {
            workspaceAPI.getTabInfo({
                tabId: response
        }).then(function(tabInfo) {
            console.log("The recordId for this tab is: " + tabInfo.recordId);
        });
        }).catch(function(error) {
            console.log(error);
        });
    }
)}
```

Controller code (recordId):

```
({
    handleOpenTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            recordId: '001xx000003DIkeAAG',
            focus: true
        }).then(function(response) {
            workspaceAPI.getTabInfo({
                  tabId: response
            }).then(function(tabInfo) {
            console.log("The url for this tab is: " + tabInfo.url);
            });
        })
        .catch(function(error) {
               console.log(error);
        });
```

```
    }
})
```

Controller code (url):

```
({
    handleOpenTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.openTab({
            url: '/lightning/r/Account/001xx000003DI05AAG/view',
            focus: true
        }).then(function(response) {
            workspaceAPI.getTabInfo({
                tabId: response
            }).then(function(tabInfo) {
            console.log("The recordId for this tab is: " + tabInfo.recordId);
            });
        }).catch(function(error) {
                console.log(error);
        });
    }
})
```

Note: The relative URL used in this example is a placeholder. To try this example yourself, use a relative URL with a record ID from your org.

Response

This method returns a promise that, upon success, resolves to the `tabId` of the new tab.

`refreshTab()` for Lightning Experience

Refreshes a workspace tab or a subtab specified by `tabId`. Keep in mind that the first subtab has the same `tabId` as the workspace tab. This method works only in Lightning console apps.

If this method is invoked for a workspace tab with unsaved changes, a confirmation window opens for the user.

- Continue editing: Changes are preserved and the tab or workspace isn't refreshed.
- Discard changes: Changes are discarded and the tab or workspace is refreshed.
- Save: Changes are saved and then the tab or workspace is refreshed.

Arguments

The method provides the same argument for both Aura Components and Lightning Web Components (LWC).

Name	Type	Description
`tabId`	string	ID of the workspace tab or subtab to refresh.
`includeAllSubtabs`	boolean	Optional. If the `tabId` corresponds to a workspace tab, all subtabs within that workspace are refreshed. The default is true. Keep in

Name	Type	Description
		mind that the first subtab has the same `tabId` as the workspace tab.

LWC Sample Code

This component checks if it's in a Lightning console app, returns information about the focused tab and refreshes it. When the getFocusedTabInfo() method resolves successfully, it returns the `tabInfo` object. The `const { tabId }` syntax destructures the `tabInfo` object and binds the `tabInfo.tabId` value on the `tabId` variable. `refreshTab()` uses this `tabId` value to refresh the tab and its subtabs.

```
import { LightningElement, wire } from 'lwc';
import {
    IsConsoleNavigation,
    getFocusedTabInfo,
    refreshTab
} from 'lightning/platformWorkspaceApi';

export default class WorkspaceAPIRefreshTab extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async refreshTab() {
        if (!this.isConsoleNavigation) {
            return;
        }
        const { tabId } = await getFocusedTabInfo();
        await refreshTab(tabId, {
            includeAllSubtabs: true
        });
    }
}
```

This example is the workspaceAPIRefreshTab component from the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, refreshes the focused workspace tab and all its open subtabs.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:workspaceAPI aura:id="workspace"/>
    <lightning:button label="Refresh Focused Tab" onclick="{!c.refreshFocusedTab}"/>
</aura:component>
```

Controller code:

```
({
    refreshFocusedTab : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
```

```
            var focusedTabId = response.tabId;
            workspaceAPI.refreshTab({
                      tabId: focusedTabId,
                      includeAllSubtabs: true
             });
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`. If there was an error, the promise is rejected.

Note: `true` doesn't necessarily mean that the refresh was successful. For example, if the tab has unsaved changes when this method was called, the user has a choice to save or discard their changes. The refresh is canceled depending on user's choice.

`removeFromBrowserTitleQueue()` for Lightning Experience

Removes a string from a list of titles that rotate in the browser title bar every three seconds. This method works only in Lightning console apps. This method isn't supported for Lightning Web Components (LWC).

Note: Accurate browser tab titles help improve accessibility. Screen readers announce page titles when a page is first loaded, and don't announce dynamic updates to the title. Use the root node of the document, like `document.title`, to announce the updated browser tab title instead.

Arguments

Name	Type	Description
`title`	string	The browser tab title to remove.

Sample Code

This component has a button that, when pressed, removes a string from a list of titles that rotate in the browser title bar every three seconds.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Remove from Browser Title Queue" onclick="{!
c.handleRemoveFromBrowserTitleQueue }" />
</aura:component>
```

Controller code:

```
({
    handleremoveFromBrowserTitleQueue : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.removeFromBrowserTitleQueue({
```

```
            title: "New Browser Title"
        })
        .then(function(result){
            console.log(result);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, open success, resolves to `true`.

`setTabHighlighted()` for Lightning Experience

Highlights the specified tab with a different background color and a badge. Tab highlights don't persist after reloading a Lightning console app. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`tabId`	string	The ID of the tab to be highlighted.
`highlighted`	boolean	Whether the tab is highlighted. Makes a utility more prominent by giving it a different background color.
`options`	object	Optional. Additional options that modify the appearance of the highlighted tab. Available options are: • `pulse`: If true, causes two colors to alternate in a smooth animation. • `state`: Changes the tab color. Available types are `success` (), `warning` (), and `error` ().

LWC Sample Code

This component checks if it's in a Lightning console app, returns information about the focused tab and highlights it in green.

```
import { LightningElement, wire } from 'lwc';
import {
    IsConsoleNavigation,
    getFocusedTabInfo,
    setTabHighlighted
} from 'lightning/platformWorkspaceApi';
```

```
export default class WorkspaceAPIHighlightTab extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async highlightTab(event) {
        if (!this.isConsoleNavigation) {
            return;
        }
        const highlighted = event.detail.checked;
        const { tabId } = await getFocusedTabInfo();
        setTabHighlighted(tabId, highlighted, {
            pulse: true,
            state: 'success'
        });
    }
}
```

This example is the workspaceAPIHighlightTab component from the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, sets the focused tab as highlighted.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Set Focused Tab Highlighted" onclick="{!
c.setFocusedTabHighlighted }" />
</aura:component>
```

Controller code:

```
({
    setFocusedTabHighlighted : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
            var focusedTabId = response.tabId;
            workspaceAPI.setTabHighlighted({
                tabId: focusedTabId,
                highlighted: true,
                options: {
                    pulse: true,
                    state: "success"
         }
            });
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, returns a `tabInfo` object representing the modified tab. A `tabInfo` object is a JSON array of information about a workspace tab, with nested arrays of information on each subtab. This is the structure of a `tabInfo` object.

```
{
     tabId: string,
     url: string (URL),
     pinned: boolean,
     closeable: boolean,
     title: string,
     icon: string (SLDS iconKey),
     iconAlt: string,
     customTitle: string (optional),
     customIcon: string (optional),
     customIconAlt: string (optional),
     highlighted: boolean,
     pageReference: object,
     isSubtab: boolean,
     parentTabId: string,
     subtabs: [
         {
             tabId: string,
             url: string (URL),
             pinned: boolean,
             closeable: boolean,
             title: string,
             icon: string (SLDS iconKey),
             iconAlt: string,
             customTitle: string (optional),
             customIcon: string (optional),
             customIconAlt: string (optional),
             highlighted: boolean,
             pageReference: object,
             isSubtab: boolean,
             parentTabId: string,
             focused: boolean,
             recordId: string,
          },
           ...
     ],
     focused: boolean,
     recordId: string
}
```

`setTabIcon()` for Lightning Experience

Sets the icon and alternative text of the specified tab. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`tabId`	string	The ID of the tab for which to set the icon.
`icon`	string	An SLDS icon key in the format `utility:iconName` where `utility` is the icon category. See a full list of utility icons on the SLDS reference site.
`iconAlt`	string	Optional. Alternative text for the icon.

LWC Sample Code

This component has a function that sets an icon on a specified tab. It uses the `tabId` property from getFocusedTabInfo().

```
import { LightningElement, wire } from 'lwc';
import {
    IsConsoleNavigation,
    getFocusedTabInfo,
    setTabIcon
} from 'lightning/platformWorkspaceApi';

export default class SetTabIconExample extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async setTabIcon() {
        if (!this.isConsoleNavigation) {
            return;
        }

        const { tabId } = await getFocusedTabInfo();
        setTabIcon(tabId, 'utility:einstein', {
            iconAlt: 'Account Insights'
        });
    }
}
```

For another example that uses `setTabIcon()`, see the workspaceAPISetTabIcon component in the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, sets the icon of the focused tab to the SLDS "Approval" action icon.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Set Focused Tab Icon" onclick="{! c.setFocusedTabIcon }" />
 </aura:component>
```

Controller code:

```
({
    setFocusedTabIcon : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
            var focusedTabId = response.tabId;
                workspaceAPI.setTabIcon({
                tabId: focusedTabId,
                icon: "action:approval",
                iconAlt: "Approval"
            });
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to a `tabInfo` object representing the modified tab. A `tabInfo` object is a JSON array of information about a workspace tab, with nested arrays of information on each subtab. This is the structure of a `tabInfo` object.

```
{
     tabId: string,
     url: string (URL),
     pinned: boolean,
     closeable: boolean,
     title: string,
     icon: string (SLDS iconKey),
     iconAlt: string,
     customTitle: string (optional),
     customIcon: string (optional),
     customIconAlt: string (optional),
     highlighted: boolean,
     pageReference: object,
     isSubtab: boolean,
     parentTabId: string,
     subtabs: [
         {
             tabId: string,
             url: string (URL),
             pinned: boolean,
             closeable: boolean,
             title: string,
             icon: string (SLDS iconKey),
             iconAlt: string,
             customTitle: string (optional),
             customIcon: string (optional),
             customIconAlt: string (optional),
             highlighted: boolean,
             pageReference: object,
```

```
            isSubtab: boolean,
            parentTabId: string,
            focused: boolean,
            recordId: string,
        },
         ...
    ],
    focused: boolean,
    recordId: string
}
```

`setTabLabel()` for Lightning Experience

Sets the label of the specified tab. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`tabId`	string	The ID of the tab for which to set the label.
`label`	string	The label of the workspace tab or subtab.

LWC Sample Code

This component has a function that sets a label on a specified tab. It uses the `tabId` property from getFocusedTabInfo().

```
import { LightningElement, wire } from 'lwc';
import {
    IsConsoleNavigation,
    getFocusedTabInfo,
    setTabLabel
} from 'lightning/platformWorkspaceApi';

const TAB_LABEL = 'Awesome Label';

export default class WorkspaceAPISetTabLabel extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async setTabLabel() {
        if (!this.isConsoleNavigation) {
            return;
        }
        const { tabId } = await getFocusedTabInfo();
        setTabLabel(tabId, TAB_LABEL);
    }
}
```

This example is the workspaceAPISetTabLabel component from the lwc-recipes repo.

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, sets the label of the focused tab to "Focused Tab".

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <lightning:button label="Set Focused Tab with Label" onclick="{! c.setFocusedTabLabel
 }" />
 </aura:component>
```

Controller code:

```
({
    setFocusedTabLabel : function(component, event, helper) {
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getFocusedTabInfo().then(function(response) {
            var focusedTabId = response.tabId;
            workspaceAPI.setTabLabel({
                tabId: focusedTabId,
                label: "Focused Tab"
            });
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to a `tabInfo` object representing the modified tab. A `tabInfo` object is a JSON array of information about a workspace tab, with nested arrays of information on each subtab. This is the structure of a `tabInfo` object.

```
{
     tabId: string,
     url: string (URL),
     pinned: boolean,
     closeable: boolean,
     title: string,
     icon: string (SLDS iconKey),
     iconAlt: string,
     customTitle: string (optional),
     customIcon: string (optional),
     customIconAlt: string (optional),
     highlighted: boolean,
     pageReference: object,
     isSubtab: boolean,
     parentTabId: string,
     subtabs: [
         {
             tabId: string,
             url: string (URL),
             pinned: boolean,
```

```
            closeable: boolean,
            title: string,
            icon: string (SLDS iconKey),
            iconAlt: string,
            customTitle: string (optional),
            customIcon: string (optional),
            customIconAlt: string (optional),
            highlighted: boolean,
            pageReference: object,
            isSubtab: boolean,
            parentTabId: string,
            focused: boolean,
            recordId: string,
        },
         ...
    ],
    focused: boolean,
    recordId: string
}
```

`EnclosingTabId` Context Wire Adapter for Lightning Experience

Returns the ID of the enclosing tab or subtab. This wire adapter is available for Lightning Web Components (LWC) only.

To determine if the component is within a tab or subtab, use this context wire adapter. If a caller component isn't using the wire adapter inside a tab or subtab, the enclosing utility tab ID is null.

Tip: To retrieve information about the tab or the subtab that a component is rendered in, first use `EnclosingTabId` instead of `getFocusedTabInfo()` on page 54. Then call `getTabInfo()` on page 57 and use the enclosing tab's ID as the argument. By using `EnclosingTabId`, you make sure that the correct tab ID is returned when you work with lifecycle hooks such as `renderedCallback()` or `connectedCallback()`.

LWC Sample Code

To get the ID of the enclosing tab in LWC, use the `EnclosingTabId` wire adapter.

This component retrieves the enclosing tab ID and closes the tab.

```
import { LightningElement, wire } from 'lwc';
import { EnclosingTabId, closeTab } from 'lightning/platformWorkspaceApi';

export class CloseEnclosingTabExample  extends LightningElement {
    @wire(EnclosingTabId) enclosingTabId;

    handleClick() {
        if (this.enclosingTabId) {
            return;
        }
        closeTab(this.enclosingTabId);
    }
}
```

For another example that uses the `EnclosingTabId` wire adapter, see openSubtab().

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Response

This method returns a promise that, upon success, resolves to the `tabId` of the enclosing tab, if within a tab. If not within a tab, this method resolves to `false` upon success.

SEE ALSO:

LWC Dev Guide: Understand the Wire Service

`IsConsoleNavigation` Context Wire Adapter for Lightning Experience

Determines whether the app it's used within uses console navigation. This wire adapter is available for Lightning Web Components (LWC) only.

To determine if the component is within a console app, use this wire adapter. If a caller component isn't using the wire adapter inside a tab or subtab, the enclosing utility tab ID is null.

LWC Sample Code

This component checks if it's within a Lightning console app and returns the tab information using the `IsConsoleNavigation` wire adapter.

```
import { LightningElement, wire } from 'lwc';
import { IsConsoleNavigation, getFocusedTabInfo } from 'lightning/platformWorkspaceApi';

export class ConsoleNavExample extends LightningElement {
    @wire(IsConsoleNavigation) isConsoleNavigation;

    async handleFocusTab() {
        if (!this.isConsoleNavigation) {
            return;
        }

        const { tabId } = await getFocusedTabInfo();
        // do something with the tabId
        });
    }
}
```

For another example that uses `IsConsoleNavigation`, see closeTab().

To make your component available for use in a Lightning console app, specify the `lightning__AppPage` target in the component's configuration file.

Response

If the current app uses console navigation, this method returns `true`, or `false` otherwise.

SEE ALSO:

LWC Dev Guide: Understand the Wire Service

Methods for the Utility Bar in Lightning Experience

The utility bar houses Aura components and Lightning web components, providing users quick access to tools they use often. The utility bar is available in Lightning Experience only. Both Lightning Web Components (LWC) and Aura Components support the utility bar methods, with usage differences noted on each method.

To understand how the utility bar methods for LWC map to the methods for Aura Components, see Utility Bar API Method Parity.

These methods are available for working with the utility bar.

IN THIS SECTION:

setPanelHeaderIcon() for Lightning Experience

Sets the icon of a utility's panel. This icon is displayed in the utility panel header. This method isn't supported for Lightning Web Components (LWC).

setPanelHeaderLabel() for Lightning Experience

Sets the label of a utility's panel. This label is displayed in the utility panel header. This method isn't supported for Lightning Web Components (LWC).

setPanelHeight() for Lightning Experience

Sets a utility panel's height. This method isn't supported for Lightning Web Components (LWC).

setPanelWidth() for Lightning Experience

Sets a utility panel's width. This method isn't supported for Lightning Web Components (LWC).

setUtilityHighlighted() for Lightning Experience

Sets a utility as highlighted, giving it a badge and a more prominent background color. This method isn't supported for Lightning Web Components (LWC).

setUtilityIcon() for Lightning Experience

Sets the icon of a utility. This icon is displayed in the utility bar. This method isn't supported for Lightning Web Components (LWC).

setUtilityLabel() for Lightning Experience

Sets the label of a utility. This text is displayed in the utility bar. This method isn't supported for Lightning Web Components (LWC).

toggleModalMode() for Lightning Experience

Toggles modal mode for a utility. While in modal mode, an overlay blocks users from using the console while the utility panel is visible. This method isn't supported for Lightning Web Components (LWC).

updatePanel() for Lightning Experience

Specifies a label and icon on the utility panel, and provides a height and width for the panel. This method is available for Lightning Web Components (LWC) only.

updateUtility() for Lightning Experience

Specifies a label and icon on the utility bar, and sets a utility as highlighted. This method is available for Lightning Web Components (LWC) only.

EnclosingUtilityId Context Wire Adapter for Lightning Experience

Determines if the component is within a utility. This wire adapter is available for Lightning Web Components (LWC) only.

`disableUtilityPopOut()` for Lightning Experience

Disables pop-out for a utility. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see enablePopout().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility to disable pop-out for. Optional when called within a utility.
`disabled`	boolean	If true, disables pop-out and removes the pop-out icon for a utility that isn't popped

Name	Type	Description
		out. If the utility is already popped out, the pop-out icon is disabled. If `disabledText` is provided, the pop-out icon isn't removed, but it's disabled.
`disabledText`	string	Hover text for the pop-out and pop-in icons if `disabled` is set to true. Optional.

Aura Components Sample Code

This component has a button that, when pressed, disables utility pop-out.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Disable Utility Pop-Out" onclick="{!
c.handleDisableUtilityPopOut }" />
</aura:component>
```

Controller code:

```
({
    handleDisableUtilityPopOut : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.disableUtilityPopOut({
            disabled: true,
            disabledText: "Pop-out is disabled"
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`enableModal()` for Lightning Experience

Toggles modal mode for a utility. While in modal mode, an overlay blocks users from using the console while the utility panel is visible. This method is available for Lightning Web Components (LWC) only.

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility for which to toggle modal mode.
`enabled`	boolean	Specifies whether to enable the utility's modal mode.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component toggles modal mode in a utility bar.

```
import { LightningElement, wire } from 'lwc';
import { enableModal, EnclosingUtilityId } from 'lightning/platformUtilityBarApi';

export default class EnableModalExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;
    isModalEnabled = false;

    async handleToggle() {
        const enable = !this.isModalEnabled;
        this.isModalEnabled = enable;
        await enableModal(this.utilityId, enable);
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

Returns a promise that resolves to `true` if successful. The promise is rejected on error.

`enablePopout()` for Lightning Experience

Toggles pop-out mode on a utility. Enabling pop-out mode on a utility displays the utility in a separate child window. This method is available for Lightning Web Components (LWC) only.

Arguments

Optional parameters are passed into an object as the last argument of the method.

Name	Type	Description
`utilityId`	string	The ID of the utility for which to toggle pop-out mode.
`enabled`	boolean	Specifies whether to enable the utility's modal mode.
`disabledText`	string	Optional. Hover text for pop-out button when the utility is not enabled for pop-out mode.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component toggles modal mode in a utility bar.

```
import { LightningElement, wire } from 'lwc';
import { enablePopout, EnclosingUtilityId } from 'lightning/platformUtilityBarApi';

export default class EnablePopoutExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;
    enable = true;

    async handleToggle() {
        const enable = !this.isPopoutEnabled;
        await enablePopout(this.utilityId, enable, { disabledText: 'disabled' });
        this.isPopoutEnabled = enable;
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

Returns a promise that resolves to `true` if successful. The promise is rejected on error.

`getAllUtilityInfo()` for Lightning Experience

Returns the state of all utilities as an array of `utilityInfo` objects.

Arguments

None.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component returns the number of utilities in the utility bar using the `utilityInfo` object.

```
import { LightningElement, wire } from 'lwc';
import { getAllUtilityInfo } from 'lightning/platformUtilityBarApi';

export default class UtilityInfoExample extends LightningElement {
    utilityCount = 0;

    async handleGetAllUtilityInfo() {
        try {
            const utilityInfo = await getAllUtilityInfo();
            this.utilityCount = utilityInfo.length;

        } catch (error) {
            // return error
        }
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Aura Components Sample Code

This component has a button that, when pressed, retrieves all `utilityInfo` objects and opens the first utility, ordered by ID.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Get All Utility Info" onclick="{! c.handleGetAllUtilityInfo
}" />
</aura:component>
```

Controller code:

```
({
    handleGetAllUtilityInfo : function(component, event, helper) {
        var utilityBarAPI = component.find("utilitybar");
        utilityBarAPI.getAllUtilityInfo().then(function(response) {
            var myUtilityInfo = response[0];
            utilityBarAPI.openUtility({
                utilityId: myUtilityInfo.id
            });
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

For both LWC and Aura Components, this method returns a promise that resolves to an array of `utilityInfo` objects, containing the following fields. The promise is rejected on error.

Name	Type	Description
`id`	string	The ID of the utility.
`isLoaded`	boolean	Whether the utility is loaded.
`utilityLabel`	string	The label of the utility.
`utilityIcon`	string	The SLDS icon ID of the utility's icon.
`utilityIconVariant`	string	The SLDS icon variant of the utility's icon.
`utilityHighlighted`	boolean	Whether the utility is highlighted.
`utilityVisible`	boolean	Whether the utility is visible.
`utilityPoppedOut`	boolean	Whether the utility is popped out.
`panelHeaderLabel`	string	The label of the utility panel.

Name	Type	Description
`panelHeaderIcon`	string	The SLDS icon ID of the utility panel's icon.
`panelHeaderIconVariant`	string	The SLDS icon variant of the utility panel's icon.
`panelHeight`	integer	The height of the utility panel in pixels.
`panelWidth`	integer	The width of the utility panel in pixels

SEE ALSO:

MDN Web Docs: async function

`getEnclosingUtilityId()` for Lightning Experience

Returns the ID of the enclosing utility, or false if not within a utility. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see EnclosingUtilityId context wire adapter.

Arguments

None.

Aura Components Sample Code

This component has a button that, when pressed, retrieves the enclosing utility's ID.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Get Enclosing Utility ID" onclick="{!
c.handleGetEnclosingUtilityId }" />
</aura:component>
```

Controller code:

```
({
    handleGetEnclosingUtilityId : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.getEnclosingUtilityId().then(function(utilityId) {
            console.log(utilityId);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to the `utilityId` of the enclosing utility or `false` if not within a utility.

`getInfo()` for Lightning Experience

Returns the state of the current utility as a `utilityInfo` object. This method is available for Lightning Web Components (LWC) only.

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility for which to retrieve the state.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component retrieves the `utilityInfo` object.

```
import { LightningElement, wire } from 'lwc';
import { EnclosingUtilityId, getInfo } from 'lightning/platformUtilityBarApi';

export default class UtilityInfoExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;

    async handleGetUtilityInfo() {
        try {
            if (!this.utilityId) {
                return;
            }
            const utilityInfo = await getInfo(this.utilityId);
            console.log(utilityInfo);
        }
        catch (error) {
            // handle error
        }
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

This method returns a promise that resolves to a `utilityInfo` object representing the enclosing utility, containing the following fields. The promise is rejected on error.

Name	Type	Description
`id`	string	The ID of the utility.
`isLoaded`	boolean	Whether the utility is loaded.
`utilityLabel`	string	The label of the utility.

Name	Type	Description
`utilityIcon`	string	The SLDS icon ID of the utility's icon.
`utilityIconVariant`	string	The SLDS icon variant of the utility's icon.
`utilityHighlighted`	boolean	Whether the utility is highlighted.
`utilityVisible`	boolean	Whether the utility is visible.
`utilityPoppedOut`	boolean	Whether the utility is popped out.
`panelHeaderLabel`	string	The label of the utility panel.
`panelHeaderIcon`	string	The SLDS icon ID of the utility panel's icon.
`panelHeaderIconVariant`	string	The SLDS icon variant of the utility panel's icon.
`panelHeight`	integer	The height of the utility panel in pixels.
`panelWidth`	integer	The width of the utility panel in pixels

SEE ALSO:

MDN Web Docs: async function

`getUtilityInfo()` for Lightning Experience

Returns the state of the current utility as a `utilityInfo` object. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see getInfo().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility for which to retrieve the state.

Aura Components Sample Code

This component has a button that, when pressed, retrieves the enclosing utility's info and opens it if it's not currently visible, and closes it otherwise.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Get Utility Info" onclick="{! c.handleGetUtilityInfo }" />
</aura:component>
```

Controller code:

```
({
    handleGetUtilityInfo : function(component, event, helper) {
        var utilityBarAPI = component.find("utilitybar");
        utilityBarAPI.getUtilityInfo().then(function(response) {
            if (response.utilityVisible) {
                utilityBarAPI.minimizeUtility();
            }
            else {
                utilityBarAPI.openUtility();
            }
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that resolves to a `utilityInfo` object representing the enclosing utility, containing the following fields. The promise is rejected on error.

Name	Type	Description
`id`	string	The ID of the utility.
`isLoaded`	boolean	Whether the utility is loaded.
`utilityLabel`	string	The label of the utility.
`utilityIcon`	string	The SLDS icon ID of the utility's icon.
`utilityIconVariant`	string	The SLDS icon variant of the utility's icon.
`utilityHighlighted`	boolean	Whether the utility is highlighted.
`utilityVisible`	boolean	Whether the utility is visible.
`utilityPoppedOut`	boolean	Whether the utility is popped out.
`panelHeaderLabel`	string	The label of the utility panel.
`panelHeaderIcon`	string	The SLDS icon ID of the utility panel's icon.
`panelHeaderIconVariant`	string	The SLDS icon variant of the utility panel's icon.
`panelHeight`	integer	The height of the utility panel in pixels.
`panelWidth`	integer	The width of the utility panel in pixels

`isUtilityPoppedOut()` for Lightning Experience

Determines whether the utility is in a popped-out state. This method isn't supported for Lightning Web Components (LWC).

To check if a utility is in a popped-out state with LWC, use getInfo() for Lightning Experience.

Arguments

None

Sample Code

This component has a button that, when pressed, states whether the current utility is popped out or not.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Is Utility popped out?" onclick="{! c.handleIsUtilityPoppedOut
 }" />
    <lightning:textarea label="Popped out?" aura:id="textarea" />
</aura:component>
```

Controller code:

```
({
    handleIsUtilityPoppedOut : function(component, event, helper) {
        var utilityBarAPI = component.find("utilitybar");
        utilityBarAPI.isUtilityPoppedOut().then(function(response) {
            component.find('textarea').set('v.value', response);
        })
        .catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true` if the utility is popped out, and false otherwise.

`minimize()` for Lightning Experience

Minimizes a utility. This method is available for Lightning Web Components (LWC) only.

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility for which to minimize.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component minimizes a utility.

```
import { LightningElement, wire } from 'lwc';
import { minimize, EnclosingUtilityId } from 'lightning/platformUtilityBarApi';

export default class MinimizeUtilityExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;

    async handleMinimize() {
        try {
            if (!this.utilityId) {
                return;
          }
        // Minimize the utility bar panel
        const isMinimized = await minimize(this.utilityId);
        console.log(`Minimize utility ${isMinimized ? 'successfully' : 'failed'}`);
        }
        catch (error) {
            // handle error
        }
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

Returns a promise that resolves to `true` if successful. The promise is rejected on error.

SEE ALSO:

LWC Dev Guide: Understand the Wire Service

`minimizeUtility()` for Lightning Experience

Minimizes a utility. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see minimize().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility to minimize. Optional when called within a utility.

Aura Components Sample Code

This component minimizes the utility when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Minimize Utility" onclick="{! c.handleMinimizeUtility }" />
</aura:component>
```

Controller code:

```
({
    handleMinimizeUtility : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.minimizeUtility();
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`onUtilityClick()` for Lightning Experience

Registers an `eventHandler` for the utility. This `eventHandler` is called when the utility is clicked.

Consider the following guidelines when working with this method.

- The method is supported in Lightning apps with standard and console navigation.
- You can use this method to register multiple callbacks per `utilityItem` when executed sequentially.
- You can't remove registered callbacks.

Arguments

The method provides the same arguments for both Aura Components and Lightning Web Components (LWC).

For LWC, optional parameters are passed into an object as the last argument of the method.

Name	Type	Description
`utilityId`	string	The ID of the utility for which to register the callback. Optional when called within a utility using Aura Components. Always required for LWC.
`eventHandler`	function	The JavaScript function that's called when the utility is clicked.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component handles a utility click using the `EnclosingUtilityId` wire adapter.

```
import { LightningElement, wire } from 'lwc';
import { onUtilityClick, EnclosingUtilityId } from 'lightning/platformUtilityBarApi';
```

```
export default class UtilityClickExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;

    handleUtilityClick() {
        if (!this.utilityId) {
            return;
        }
        onUtilityClick(this.utilityId, () => {
            console.log(`Utility ${this.utilityId} clicked!`);
        });
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

LWC Response

Returns a promise that resolves to `true` if successful. The promise is rejected on error.

Aura Components Sample Code

This component has a button that, when pressed, registers an `eventHandler` function for the enclosing utility.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global">
    <lightning:utilityBarAPI aura:id="utilitybar" />
   <lightning:button label="Register Event Handler" onclick="{!c.handleOnUtilityClick}"/>
</aura:component>
```

Controller code:

```
({
  handleOnUtilityClick: function(component, event, helper){
        var utilityBarAPI = component.find("utilitybar");
   var eventHandler = function(response){
            console.log(response);
        };

        utilityBarAPI.onUtilityClick({
               eventHandler: eventHandler
        }).then(function(result){
            console.log(result);
        }).catch(function(error){
         console.log(error);
        });
    }
})
```

Aura Components Response

This method returns a promise that, upon success, resolves to `true`, and is rejected on error. The `eventHandler` expects a response JSON object containing the following fields.

Name	Type	Description
`utilityId`	string	The ID of the utilityBar item button that was clicked.
`panelVisible`	boolean	False if the utility item panel is hidden after the button is clicked. True if the utility item panel is visible after the button is clicked.

`open()` for Lightning Experience

Opens a utility. If the utility is already open, this method has no effect. Only one utility can be open at a time. If another utility is already open, `open()` minimizes the utility. This method is available for Lightning Web Components (LWC) only.

Arguments

Optional parameters are passed into an object as the last argument of the method.

Name	Type	Description
`utilityId`	string	The ID of the utility to open.
`autoFocus`	object	Optional. Specifies whether the utility item to open has focus.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component opens a utility using the enclosing utility ID.

```
import { LightningElement, wire } from 'lwc';
import { open, EnclosingUtilityId } from 'lightning/platformUtilityBarApi';

export default class EnablePopoutExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;

    async handleOpen() {
        if (!this.utilityId) {
            return;
        }
        await open(this.utilityId, { autoFocus: true });
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

Returns a promise that resolves to `true` if successful. The promise is rejected on error.

`openUtility()` for Lightning Experience

Opens a utility. If the utility is already open, this method has no effect. Only one utility can be open at a time. If another utility is already open, `openUtility()` minimizes the utility. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see open().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility to open. Optional when called within a utility.

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, opens the utility when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Open Utility" onclick="{! c.handleOpenUtility }" />
</aura:component>
```

Controller code:

```
({
    handleOpenUtility : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.openUtility();
    }
})
```

This example opens a utility from outside of the utility, using the `utilityId` field.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Open First Utility" onclick="{! c.openFirstUtility }" />
</aura:component>
```

Controller code:

```
({
    openFirstUtility : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.getAllUtilityInfo().then(function(response) {
            var myUtilityInfo = response[0];
            utilityAPI.openUtility({
                utilityId: myUtilityInfo.id
            });
        })
        .catch(function(error) {
            console.log(error);
```

```
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`setPanelHeaderIcon()` for Lightning Experience

Sets the icon of a utility's panel. This icon is displayed in the utility panel header. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see updatePanel().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility to set the panel header icon on. Optional when called within a utility.
`icon`	string	An SLDS utility icon key. This is displayed in the utility bar. See a full list of utility icon keys on the SLDS reference site.
`options`	object	Optional. Additional options that modify the appearance of the utility panel icon. • `iconVariant`—Changes the utility panel icon color. Available types are `success` ( ), `warning` ( ), and `error` ( ).

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, sets the icon of the utility panel to a yellow SLDS "frozen" icon when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Set Panel Header Icon" onclick="{! c.handleSetPanelHeaderIcon
 }" />
</aura:component>
```

Controller code:

```
({
    handleSetPanelHeaderIcon : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
```

```
        utilityAPI.setPanelHeaderIcon({
            icon: "frozen"
            options:{
                iconVariant:"warning"
            }
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`setPanelHeaderLabel()` for Lightning Experience

Sets the label of a utility's panel. This label is displayed in the utility panel header. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see updatePanel().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility to set the panel header label on. Optional when called within a utility.
`label`	string	The label of the utility displayed in the panel header.

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, sets the label of the utility panel to "My Utility" when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
   <lightning:button label="Set Panel Header Label" onclick="{! c.handleSetPanelHeaderLabel
 }" />
</aura:component>
```

Controller code:

```
({
    handleSetPanelHeaderLabel : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.setPanelHeaderLabel({
            label: "My Utility"
        });
```

```
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`setPanelHeight()` for Lightning Experience

Sets a utility panel's height. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see updatePanel().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility of which to set the height. Optional when called within a utility.
`heightPX`	integer	The height of the utility panel in pixels.

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, sets the height of the utility to 500 pixels when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Set Panel Height" onclick="{! c.handleSetPanelHeight }" />
</aura:component>
```

Controller code:

```
({
    handleSetPanelHeight : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.setPanelHeight({
            heightPX: 500
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`setPanelWidth()` for Lightning Experience

Sets a utility panel's width. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see updatePanel().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility of which to set the width. Optional when called within a utility.
`widthPX`	integer	The width of the utility panel in pixels.

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, sets the width of the utility panel to 800 pixels when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Set Panel Width" onclick="{! c.handleSetPanelWidth }" />
</aura:component>
```

Controller code:

```
({
    handleSetPanelWidth : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.setPanelWidth({
            widthPX: 800
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`setUtilityHighlighted()` for Lightning Experience

Sets a utility as highlighted, giving it a badge and a more prominent background color. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see updateUtility() for Lightning Experience.

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility to highlight. Optional when called within a utility.

Name	Type	Description
`highlighted`	boolean	Whether the utility is highlighted. Makes a utility more prominent by giving it a different background color.

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, sets a utility as highlighted when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Set Utility Highlighted" onclick="{!
c.handleSetUtilityHighlighted}" />
</aura:component>
```

Controller code:

```
({
    handleSetUtilityHighlighted : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.setUtilityHighlighted({
            highlighted: true
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`setUtilityIcon()` for Lightning Experience

Sets the icon of a utility. This icon is displayed in the utility bar. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see updateUtility() for Lightning Experience.

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility on which to set the icon. Optional when called within a utility.
`icon`	string	An SLDS utility icon key that is displayed in the utility bar. See a full list of utility icon keys on the SLDS reference site.

Name	Type	Description
`options`	object	Optional. Additional options that modify the appearance of the utility icon. • `iconVariant`—Changes the utility icon color. Available types are `success` (), `warning` (), and `error` ().

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, sets the icon of the utility to a green SLDS "insert_tag_field" icon when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Set Utility Icon" onclick="{! c.handleSetUtilityIcon }" />
</aura:component>
```

Controller code:

```
({
    handleSetUtilityIcon : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.setUtilityIcon({
            icon: "insert_tag_field",
            options:{
                iconVariant:"success"
            }
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`setUtilityLabel()` for Lightning Experience

Sets the label of a utility. This text is displayed in the utility bar. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see updateUtility() for Lightning Experience.

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility of which to set the label. Optional when called within a utility.

Name	Type	Description
`label`	string	The label of the utility displayed in the utility bar.

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, sets the label of the utility to "My Favorite Utility" when the button is pressed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Set Utility Label" onclick="{! c.handleSetUtilityLabel }" />
</aura:component>
```

Controller code:

```
({
    handleSetUtilityLabel : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.setUtilityLabel({
            label: "My Favorite Utility"
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`toggleModalMode()` for Lightning Experience

Toggles modal mode for a utility. While in modal mode, an overlay blocks users from using the console while the utility panel is visible. This method isn't supported for Lightning Web Components (LWC).

For LWC usage, see enableModal().

Arguments

Name	Type	Description
`utilityId`	string	The ID of the utility to open. Optional when called within a utility.
`enableModalMode`	boolean	Whether to enable modal mode.

Aura Components Sample Code

This component, when added to a single-column Lightning page used in a utility bar, has a button that, when pressed, toggles modal mode.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:utilityBarAPI aura:id="utilitybar" />
    <lightning:button label="Toggle Modal Mode" onclick="{! c.handleToggleModalMode }" />
</aura:component>
```

Controller code:

```
({
    handleToggleModalMode : function(component, event, helper) {
        var utilityAPI = component.find("utilitybar");
        utilityAPI.toggleModalMode({
            enableModalMode: true
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true`.

`updatePanel()` for Lightning Experience

Specifies a label and icon on the utility panel, and provides a height and width for the panel. This method is available for Lightning Web Components (LWC) only.

Arguments

Optional parameters are passed into an object as the last argument of the method.

Name	Type	Description
`utilityId`	string	The ID of the utility panel to update.
`panelAttrs`	object	Optional. Attributes that specify the appearance of the panel. • `label`—The label of the utility that's displayed in the panel header. • `icon`—The Lightning Design System (SLDS) name of the icon in the format `einstein`, where `einstein` is the name of the SLDS utility icon. The icon is displayed in the utility panel header. See a full list of utility icon keys at the SLDS reference site. SLDS doctype, standard, custom, and action icons aren't supported. • `iconVariant`—The variant changes the color of the utility panel icon.

Name	Type	Description
		Available variants are `success` (), `warning` (), and `error` (). • `height`—The height of the utility panel in pixels. • `heightTransition`—An object that describes the CSS transition for the utility panel's height. See the **heightTransition** section. • `width`—The width of the utility panel in pixels. • `widthTransition`—An object that describes the CSS transition for the utility panel's width. See the **widthTransition** section.

heightTransition

An optional object that describes the CSS transition for the utility panel's height. Pass this object to `panelAttrs` and use with `height`. Optional arguments include:

- `durationMs`—The time in milliseconds. It takes for the height transition to complete. The default value is 0.
- `timingFunction`—The transition timing function that sets the rate for panel height changes. Applies any CSS easing function that's supported by your target browser. The default value is `ease`.
- `delayMs`—The wait time in milliseconds before the height transition starts. The default value is 0.

widthTransition

An optional object that describes the CSS transition for the utility panel's width. Pass this object to `panelAttrs` and use with `width`. Optional arguments include:

- `durationMs`—The time in milliseconds. It takes for the width transition to complete. The default value is 0.
- `timingFunction`—The transition timing function that sets the rate for panel width changes. Applies any CSS easing function that's supported by your target browser. The default value is `ease`.
- `delayMs`—The wait time in milliseconds before the width transition starts. The default value is 0.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component opens a utility using the enclosing utility ID.

```
import { LightningElement, wire } from 'lwc';
import { updatePanel, EnclosingUtilityId } from 'lightning/platformUtilityBarApi';

export default class UpdatePanelExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;
```

```
    panelAttrs = {
        label: 'Account Insights',
        icon: 'einstein',
        iconVariant: 'success',
        height: 600,
        width: 600
    }

    handleUpdate() {
        if (this.utilityId) {
            return updatePanel(this.utilityId, this.panelAttrs);
        }
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

Returns a promise that resolves to `true` if successful. The promise is rejected on error.

`updateUtility()` for Lightning Experience

Specifies a label and icon on the utility bar, and sets a utility as highlighted. This method is available for Lightning Web Components (LWC) only.

Arguments

Optional parameters are passed into an object as the last argument of the method.

Name	Type	Description
`utilityId`	string	The ID of the utility to update.
`utilityAttrs`	object	Optional. Attributes that specify the appearance of the icon. • `label`—The label of the utility that's displayed in the utility bar. • `icon`—The Lightning Design System (SLDS) name of the icon in the format `einstein`, where `einstein` is the name of the SLDS utility icon. The icon is displayed in the utility bar. See a full list of utility icon keys at the SLDS reference site. SLDS doctype, standard, custom, and action icons aren't supported. • `iconVariant`—The variant changes the color of the utility bar. Available

Name	Type	Description
		variants are `success` (), `warning` (), and `error` (). • `highlighted`—Applies a different background color.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component opens a utility using the enclosing utility ID.

```
import { LightningElement, wire } from 'lwc';
import { updateUtility, EnclosingUtilityId } from 'lightning/platformUtilityBarApi';

export default class UpdatePanelExample extends LightningElement {
    @wire(EnclosingUtilityId) utilityId;

    utilityAttrs = {
        label: 'Account Insights',
        icon: 'einstein',
        iconVariant: 'success',
        highlighted: true
    }

    handleUpdate() {
        if (this.utilityId) {
            return updateUtility(this.utilityId, this.utilityAttrs);
        }
    }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

Returns a promise that resolves to `true` if successful. The promise is rejected on error.

`EnclosingUtilityId` Context Wire Adapter for Lightning Experience

Determines if the component is within a utility. This wire adapter is available for Lightning Web Components (LWC) only.

To obtain the ID of the enclosing utility, use this context wire adapter. If a caller component isn't using the wire adapter inside a panel, the enclosing utility ID is null.

LWC Sample Code

Before you can use the LWC utility bar API, Lightning Web Security must be enabled.

This component minimizes a utility bar panel using the enclosing utility ID and the minimize() method.

```
import { LightningElement, wire } from 'lwc';
import { EnclosingUtilityId, minimizeUtility } from 'lightning/platformUtilityBarApi';

export default class MinimizeUtilityExample extends LightningElement {
  @wire(EnclosingUtilityId) utilityId;

  handleClick() {
    if (!this.utilityId) {
      return;
    }
    // Minimize the utility bar panel
    minimize(this.utilityId);
  }
}
```

To make your component available for use in a utility bar, specify the `lightning__UtilityBar` target in the component's configuration file.

Response

Returns the enclosing utility ID if the caller component is within a utility, or `null` otherwise.

SEE ALSO:

LWC Dev Guide: Understand the Wire Service

LWC Methods for Enhanced Messaging in Lightning Experience

The Conversation Toolkit API for Enhanced Messaging provides methods to interact with a Messaging customer from a Lightning web component (LWC). These methods apply to Lightning web components in Lightning Experience only.

The Conversation Toolkit API is designed for Enhanced Messaging channels, which includes Messaging for In-App and Web, Enhanced Apple, Enhanced Facebook Messenger, Enhanced WhatsApp, Enhanced SMS, and Partner Messaging.

You can now import Lightning console API methods, which allow your background and non-rendered custom components to use the APIs.

These methods help developers customize the agent experience and how users and other components interact with the conversation component on a page. For example, if you want to customize how an agent composes a message, you can create a messaging composer to draft and send a message during the conversation. These methods only work with an open Messaging Session record page in the console or standard app. If the record is not open, the methods don't work.

Note: Use only rendered components with the Conversation Toolkit APIs. If you use a component that doesn't have markup or that operates in the background, the APIs don't work. The conversation component must also be rendered for the APIs to work.

Sample Code

This sample code is an example of the .html file of an LWC bundle that uses Conversation Toolkit API.

```
<template>
    <lightning-card title="LWC tool kit api" icon-name="custom:custom14">
      <lightning-conversation-toolkit-api lwc:ref="lwcToolKitApi">
      </lightning-conversation-toolkit-api>
```

```
      <div>
        {apiOutput}
      </div>
      <div>
        <lightning-button label="getConversationLog" onclick={handleButtonClick}
value="getConversationLog"></lightning-button>
        <lightning-button label="sendTextMessage" onclick={handleButtonClick}
value="sendTextMessage"></lightning-button>
        <lightning-button label="setAgentInput" onclick={handleButtonClick}
value="setAgentInput"></lightning-button>
        <lightning-button label="endConversation" onclick={handleButtonClick}
value="endConversation"></lightning-button>
      </div>
      <template for:each={log} for:item="item">
        <li key={item}>
          {item}
        </li>
      </template>

    </lightning-card>
  </template>
```

This sample code is an example of the .xml metadata file of the LWC bundle.

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

  <apiVersion>52.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__AppPage</target>
    <target>lightning__RecordPage</target>
    <target>lightning__HomePage</target>
  </targets>
</LightningComponentBundle>
```

This sample code is an example of the .js file of the LWC bundle. Here is where you use the LWC methods for Enhanced Messaging.

```
import { LightningElement, api,track } from 'lwc';

export default class HelloWorld extends LightningElement {
  @api recordId;

  @track log = [];
  @track apiOutput;
  changeHandler(event) {
    this.greeting = event.target.value;
  }

  async handleButtonClick(event){
    this.reset();
    const toolKit = this.refs.lwcToolKitApi;
    let result;
    switch (event.target.value) {
      case 'getConversationLog':
```

```
        result = await toolKit.getConversationLog(this.recordId);
        for(let i=0;i<result.messages.length;i++){
          var msg = {
              type:result.messages[i].type,
              content:result.messages[i].content,
              name:result.messages[i].name,
              timestamp:result.messages[i].timestamp
          }
          this.log.push(JSON.stringify(msg));
        }
        break;
      case 'sendTextMessage':
        result = await toolKit.sendTextMessage(this.recordId, { text: 'Text from toolkit'
});
        break;
      case 'setAgentInput':
       result = await toolKit.setAgentInput(this.recordId,{ text: 'Inserting from toolkit'
});
        break;
      case 'endConversation':
        result = await toolKit.endConversation(this.recordId);
        break;
    }
    if(result){
      this.apiOutput = event.target.value + " sucess";
    } else {
      this.apiOutput = event.target.value + " failed";
    }

  }

  reset(){
    this.log = [];
    this.apiOutput='';
  }

}
```

These are the LWC methods for Enhanced Messaging.

IN THIS SECTION:

setMessagingComponent() for LWC for Lightning Experience

Inserts a specified messaging component into the service rep's text box. You can also use this method to insert text into the service rep's text box. This method works only in Lightning console apps.

sendMessagingComponent() for LWC for Lightning Experience

Sends a new message with a specified messaging component on behalf of the service rep. You can also use this method to send a text message. This method works only in Lightning console apps.

SEE ALSO:

Salesforce Trailhead: Build Lightning Web Components

Salesforce Help: Customize the Enhanced Conversation Component in the Agent Console

Salesforce Help: Messaging Component Types and Formats

endConversation() for LWC for Lightning Experience

Ends the Messaging session. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`recordId`	String	The ID of the record for the Messaging session.

Response

Returns a Promise. Success resolves to `true`. The Promise is rejected if there's an error.

getConversationLog() for LWC for Lightning Experience

Retrieves the conversation log. This method works only in Lightning console apps.

This method returns up to 200 entries from the point the user scrolled to in the transcript. Rich content from bots, for example, messaging components with a question with choices, are not supported.

Arguments

Name	Type	Description
`recordId`	String	The ID of the record for the Messaging session.

Response

Returns a Promise. Success resolves to a `messages` response object. The Promise is rejected if there's an error.

Name	Type	Description
`messages`	Array of `message` objects.	An array of message objects containing all of the messages from the conversation log.

message

The message object containing a single message from the conversation log.

Name	Type	Description
`content`	String	The text content of a message in the conversation log.
`name`	String	The name of the user who sent the message in the conversation log. This name appears exactly as it is displayed in the conversation.
`type`	String	The type of message that was received, such as `Agent` or `EndUser`.
`timestamp`	String	The date and time the message was received.

sendTextMessage() for LWC for Lightning Experience

Sends a new text message from the agent to an end user. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`recordId`	String	The ID of the record for the Messaging session.
`message`	Object	The `message` object with the message to send to the customer.

`message`

Name	Type	Description
`text`	String	The message to the customer. For example: `{ text: "This is a sample message." }`

Response

Returns a Promise. Success resolves to `true`. The Promise is rejected if there's an error.

setAgentInput() for LWC for Lightning Experience

Sets the text in the agent's text box. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`recordId`	String	The ID of the record for the Messaging session.
`message`	Object	The `message` object with the message to place in the agent's text box.
`setAtCursor`	Boolean	Optional. Indicates whether to insert the message at the current cursor location. If `false`, the message overwrites any existing text. Default value is `false`.

`message`

Name	Type	Description
`text`	String	The message to the agent. For example: `{ text: "This is a sample message." }`

Response

Returns a Promise. Success resolves to `true`. The Promise is rejected if there's an error.

setMessagingComponent() for LWC for Lightning Experience

Inserts a specified messaging component into the service rep's text box. You can also use this method to insert text into the service rep's text box. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`recordId`	String	The ID of the record for the Messaging session.
messageType	String	The message component type to place in the service rep's text box. For example, StaticContentMessage. See Message Types and Message Format Types.
nameOrId	String	The name or ID of the messaging component to place in the service rep's text box.
text	String	Optional. The message to the end user. For example: `This is a sample message.`
setAtCursor	Boolean	Optional. Indicates whether to insert the message at the current cursor location. If `false`, the message overwrites any existing text. Default value is `false`.

Response

Returns a Promise. Success resolves to `true`. The Promise is rejected if there's an error.

Sample Code

This example inserts a static content message into the service rep's text box.

```
import { LightningElement, api } from 'lwc';
import { setMessagingComponent } from 'lightning/conversationToolkitApi';

export default class MyComponent extends LightningElement {
    @api recordId;

    async handleButtonClick(event) {
        const result = await setMessagingComponent(
            this.recordId, {
                messageType: "StaticContentMessage",
                nameOrId: "1mdxx0000000001AAA"
            });
        console.log(result);
    }
}
```

sendMessagingComponent() for LWC for Lightning Experience

Sends a new message with a specified messaging component on behalf of the service rep. You can also use this method to send a text message. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`recordId`	String	The ID of the record for the Messaging session.
messageType	String	The message component type to send on behalf of the service rep. For example, `StaticContentMessage`. See Message Types and Message Format Types.
nameOrId	String	The name or ID of the messaging component to send on behalf of the service rep.
text	String	Optional. The message to the end user. For example: `This is a sample message.`

Response

Returns a Promise. Success resolves to `true`. The Promise is rejected if there's an error.

Sample Code

This example sends a static content message on behalf of the service rep.

```
import { LightningElement, api } from 'lwc';
import { sendMessagingComponent } from 'lightning/conversationToolkitApi';

export default class MyComponent extends LightningElement {
```

```
    @api recordId;

    async handleButtonClick(event) {
        const result = await setMessagingComponent(
            this.recordId, {
                messageType: "StaticContentMessage",
                nameOrId: "1mdxx0000000001AAA"
            });
        console.log(result);
    }
}
```

Aura Methods for Enhanced Messaging in Lightning Experience

Enable your developers to customize the agent experience by allowing custom components to interact with the Enhanced Conversation Component. These methods apply to Aura components in Lightning Experience only.

Use Enhanced Messaging methods when:

- The methods are invoked within the page context of the Enhanced Messaging session.
- The Enhanced Messaging session is active.
- The Enhanced Conversation Component is visible on the page.

These methods apply to Aura components in Lightning Experience only.

IN THIS SECTION:

`endChat()` for Lightning Experience

Ends a chat in which an agent is currently engaged. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the chat that you want to end.

Sample Code

This example ends the chat and logs the result.

Component Code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <aura:attribute name="recordId" type="String" />
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <ui:button label="endChat" press="{!c.endChat}" />
</aura:component>
```

Controller Code:

```
({
    endChat: function(cmp, evt, helper) {
        var conversationKit = cmp.find("conversationKit");
        var recordId = cmp.get("v.recordId");
        conversationKit.endChat({
            recordId: recordId
        })
        .then(function(result){
            if (result) {
                    console.log("Successfully ended chat");
                } else {
                    console.log("Failed to end chat");
                }
        });
    }
})
```

Response

Returns a Promise. Success resolves to true. The Promise is rejected if there's an error.

`getChatLog()` for Lightning Experience

Returns the chat log of an Enhanced Messaging chat associated with a specific recordId. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the work associated with the current chat.

Sample Code

This example retrieves the chat log for the given chat, logs the result, and if successful, saves the result to a variable.

Component Code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <aura:attribute name="recordId" type="String" />
  <aura:attribute name="chatLog" type="Object" />
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <ui:button label="getChatLog" press="{!c.getChatLog}" />
</aura:component>
```

Controller Code:

```
({
    getChatLog: function(cmp, evt, helper) {
        var conversationKit = cmp.find("conversationKit");
        var recordId = cmp.get("v.recordId");
        conversationKit.getChatLog({
            recordId: recordId
        })
        .then(function(result){
            if (result) {
                    console.log("Successfully retrieved chat log");
                    cmp.set("v.chatLog", result);
                } else {
                    console.log("Failed to retrieve chat log");
                }
        });
    }
})
```

Response

Returns a Promise. Success resolves to a response object containing the `messages`. The Promise is rejected if there's an error.

Name	Type	Description
`messages`	Array of `message` objects.	An array of chat message objects containing all of the chat messages from the chat log.

`message`

The `message` object contains a single chat message from the chat log and the following properties:

Property	Type	Description
`content`	String	The text content of a message in the chat log.
`name`	String	The name of the user who sent the message in the chat log. This name appears exactly as it is displayed in the chat log.
`type`	String	The type of message that was received, such as Agent or Visitor.
`timestamp`	Date/Time	The date and time the chat message was received.

`sendMessage()` for Lightning Experience

Sends a new chat message from the agent to a chat with a specific chat key. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the chat that you want to end.
`message`	Object	An object containing the data to send in the message.

`message`

Name	Type	Description
`text`	String	The text to be sent in the message.

Sample Code

This example sends a message to the visitor and logs the result.

Component Code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <aura:attribute name="recordId" type="String" />
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <ui:button label="sendMessage" press="{!c.sendMessage}" />
</aura:component>
```

Controller Code:

```
({
    sendMessage: function(cmp, evt, helper) {
        var conversationKit = cmp.find("conversationKit");
        var recordId = cmp.get("v.recordId");
        conversationKit.sendMessage({
            recordId: recordId,
            message: {
                text:"Hi, this was sent using the sendMessage API!"
            }
        })
        .then(function(result){
            if (result) {
                    console.log("Successfully sent message");
                } else {
                    console.log("Failed to send message");
                }
        });
    }
})
```

Response

Returns a Promise. Success resolves to true. The Promise is rejected if there's an error.

`setAgentInput()` for Lightning Experience

Sets the text in the agent's text box while showing typing indicators. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the work associated with the current chat.
`message`	Object	The message to place in the agent's text box. The message should be an object with a string property for the text value. For example: `{ text: "This is a sample message." }`
`setAtCursor`	Boolean	Indicates whether to insert the message at the current cursor location. If `false`, the message overwrites any existing text. Default value is `false`.

Response

Returns a Promise. Success resolves to true. The Promise is rejected if there's an error.

Methods for Chat in Lightning Experience

Let customers chat with your agents on your web page.

Connect with customers or website visitors in real time through Web-based chat.

Important: The legacy chat product is in maintenance-only mode, and we won't continue to build new features. You can continue to use it, but we no longer recommend that you implement new chat channels. Instead, you can modernize your customer communication with Messaging for In-App and Web. Messaging offers many of the chat features that you love plus asynchronous conversations that can be picked back up at any time. For Lightning Console JavaScript API, use Aura Methods for Enhanced Messaging in Lightning Experience.

IN THIS SECTION:

`endChat()` for Lightning Experience

Ends a chat in which an agent is currently engaged. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the chat that you want to end.

Sample Code

This example ends the chat and logs the result.

Component Code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <aura:attribute name="recordId" type="String" />
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <ui:button label="endChat" press="{!c.endChat}" />
</aura:component>
```

Controller Code:

```
({
    endChat: function(cmp, evt, helper) {
        var conversationKit = cmp.find("conversationKit");
        var recordId = cmp.get("v.recordId");
        conversationKit.endChat({
            recordId: recordId
        })
        .then(function(result){
            if (result) {
                    console.log("Successfully ended chat");
                } else {
                    console.log("Failed to end chat");
                }
        });
    }
})
```

Response

Returns a Promise. Success resolves to true. The Promise is rejected if there's an error.

`getChatLog()` for Lightning Experience

Returns the chat log of a chat associated with a specific recordId. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the work associated with the current chat.

Sample Code

This example retrieves the chat log for the given chat, logs the result, and if successful, saves the result to a variable.

Component Code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <aura:attribute name="recordId" type="String" />
  <aura:attribute name="chatLog" type="Object" />
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <ui:button label="getChatLog" press="{!c.getChatLog}" />
</aura:component>
```

Controller Code:

```
({
    getChatLog: function(cmp, evt, helper) {
        var conversationKit = cmp.find("conversationKit");
        var recordId = cmp.get("v.recordId");
        conversationKit.getChatLog({
            recordId: recordId
        })
        .then(function(result){
            if (result) {
                    console.log("Successfully retrieved chat log");
                    cmp.set("v.chatLog", result);
                } else {
                    console.log("Failed to retrieve chat log");
                }
        });
    }
})
```

Response

Returns a Promise. Success resolves to a response object containing the `messages` and `customEvents` properties. The Promise is rejected if there's an error.

Name	Type	Description
`customEvents`	Array of `customEvent` objects.	An array of custom event objects representing the custom events that occurred during a chat.
`messages`	Array of `message` objects.	An array of chat message objects containing all of the chat messages from the chat log.
`success`	Boolean	`true` if getting the chat log was successful; `false` if getting the chat log wasn't successful.

`customEvent`

The `customEvent` object contains a single event from the chat log and the following properties:

Property	Type	Description
`source`	String	The person who initiated the custom event, either the chat visitor or the agent.
`type`	String	The type of custom event that occurred.
`data`	String	The data of the custom event that was sent to the chat; corresponds to the `data` argument of the `liveagent.chasitor.sendCustomEvent()` method used to send this event from the chat window.
`timestamp`	Date/Time	The date and time a custom event was received.

`message`

The `message` object contains a single chat message from the chat log and the following properties:

Property	Type	Description
`content`	String	The text content of a message in the chat log.
`name`	String	The name of the user who sent the message in the chat log. This name appears exactly as it is displayed in the chat log.
`type`	String	The type of message that was received, such as Agent or Visitor.
`timestamp`	Date/Time	The date and time the chat message was received.

`sendCustomEvent()` for Lightning Experience

Sends a custom event to the client-side chat window for a chat with a specific chat key. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the event that you want to customize.
`type`	String	The name of the custom event type.
`data`	String	The data attached to the custom event.

Sample Code

This example publishes a custom event and logs the result.

Component Code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <aura:attribute name="recordId" type="String" />
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <ui:button label="sendCustomEvent" press="{!c.sendCustomEvent}" />
</aura:component>
```

Controller Code:

```
({
    sendCustomEvent: function(cmp, evt, helper) {
        var conversationKit = cmp.find("conversationKit");
        var recordId = cmp.get("v.recordId");
        var type = "myCustomEventType";
        var data = "myCustomEventData";
        conversationKit.sendCustomEvent({
            recordId: recordId,
            type: type,
            data: data
        })
        .then(function(result){
            if (result) {
                    console.log("Successfully sent custom event");
                } else {
                    console.log("Failed to send custom event");
                }
        });
    }
})
```

The custom event type must match the name of your custom event. Replace `myCustomEventType` with your own custom event name.

Response

Returns a Promise. Success is indicated if the promise is fulfilled. Failure is indicated if the catch clause is invoked.

`sendMessage()` for Lightning Experience

Sends a new chat message from the agent to a chat with a specific chat key. This method works only in Lightning console apps.

Arguments

Name	Type	Description
`argumentObj`	Object	An object containing all the arguments to be passed into this method.

`argumentObj`

Name	Type	Description
`recordId`	String	The ID of the chat that you want to end.
`message`	Object	An object containing the data to send in the message.

`message`

Name	Type	Description
`text`	String	The text to be sent in the message.

Sample Code

This example sends a message to the visitor and logs the result.

Component Code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <aura:attribute name="recordId" type="String" />
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <ui:button label="sendMessage" press="{!c.sendMessage}" />
</aura:component>
```

Controller Code:

```
({
    sendMessage: function(cmp, evt, helper) {
        var conversationKit = cmp.find("conversationKit");
        var recordId = cmp.get("v.recordId");
        conversationKit.sendMessage({
            recordId: recordId,
            message: {
                text:"Hi, this was sent using the sendMessage API!"
            }
        })
        .then(function(result){
            if (result) {
                    console.log("Successfully sent message");
                } else {
                    console.log("Failed to send message");
                }
        });
    }
})
```

Response

Returns a Promise. Success resolves to true. The Promise is rejected if there's an error.

Methods for Omni-Channel in Lightning Experience

Omni-Channel lets your call center route any type of incoming work item to the most qualified, available agents.

For more information about Omni-Channel, see *Omni-Channel for Administrators* in Salesforce Help.

IN THIS SECTION:

`acceptAgentWork` for Lightning Experience

Accepts a work item that's assigned to an agent.

Arguments

Name	Type	Description
`workId`	string	The ID of the work item the agent accepts.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Accept" onclick="{! c.acceptWork }" />
</aura:component>
```

Controller code:

```
({
    acceptWork: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.getAgentWorks().then(function(result) {
            var works = JSON.parse(result.works);
            var work = works[0];
            omniAPI.acceptAgentWork({workId: work.workId}).then(function(res) {
                if (res) {
                    console.log("Accepted work successfully");
                } else {
                    console.log("Accept work failed");
                }
            }).catch(function(error) {
                console.log(error);
            });
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true` and is rejected on error.

`closeAgentWork` for Lightning Experience

Changes the status of a work item to *Closed* and removes it from the list of work items in the Omni-Channel utility.

Arguments

Name	Type	Description
`workId`	string	The ID of the work item that's closed.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Close" onclick="{! c.closeWork }" />
</aura:component>
```

Controller code:

```
({
    closeWork: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.getAgentWorks().then(function(result) {
            var works = JSON.parse(result.works);
            var work = works[0];
            omniAPI.closeAgentWork({workId: work.workId}).then(function(res) {
                if (res) {
                    console.log("Closed work successfully");
                } else {
                    console.log("Close work failed");
                }
            }).catch(function(error) {
                console.log(error);
            });
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true` and is rejected on error.

`declineAgentWork` for Lightning Experience

Declines a work item that's assigned to an agent.

Arguments

Name	Type	Description
`workId`	string	The ID of the work item that the agent declines.
`declineReason`	string	The reason that the agent declined the work request.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Decline" onclick="{! c.declineWork }" />
</aura:component>
```

Controller code:

```
({
    declineWork: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.getAgentWorks().then(function(result) {
            var works = JSON.parse(result.works);
```

```
            var work = works[0];
            omniAPI.declineAgentWork({workId: work.workId}).then(function(res) {
                if (res) {
                    console.log("Declined work successfully");
                } else {
                    console.log("Decline work failed");
                }
            }).catch(function(error) {
                console.log(error);
            });
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true` and is rejected on error.

`getAgentWorkload for Lightning Experience`

Retrieves an agent's currently assigned workload. Use this method to reroute work to available agents.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Get workload" onclick="{! c.getAgentWorkload }" />
</aura:component>
```

Controller code:

```
({
    getAgentWorkload: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.getAgentWorkload().then(function(result) {
            console.log('Retrieved Agent Configured Capacity and Current Workload
successfully');
            console.log('Agent\'s configured capacity is: ' + result.configuredCapacity);

            console.log('Agent\'s currently assigned workload is: ' +
result.currentWorkload);
        }).catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to an object containing the following fields.

Name	Type	Description
`configuredCapacity`	number	The agent's configured primary capacity (work that's assigned to the current user) through Presence Configuration.
`currentWorkload`	number	The agent's currently assigned primary workload.
`configuredInterruptibleCapacity`	Number	Indicates the agent's configured interruptible capacity (that is, work that's assigned to the current user) through Presence Configuration.
`currentInterruptibleWorkload`	Number	Indicates the agent's currently assigned interruptible workload.

`getAgentWorks` for Lightning Experience

Returns a list of work items that are assigned to an agent and open in the agent's workspace.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Get Agent works" onclick="{! c.getAgentWorks }" />
</aura:component>
```

Controller code:

```
({
    getAgentWorks: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.getAgentWorks().then(function(result) {
            var works = JSON.parse(result.works);
            console.log('First Agent Work ID is: ' +  works[0].workId);
            console.log('Assigned Entity Id of the first Agent Work is: ' +
works[0].workItemId);
            console.log('Is first Agent Work Engaged: ' + works[0].isEngaged);
        }).catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to an array of `work` objects, containing the following fields.

Name	Type	Description
`workItemId`	String	The ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent.
`workId`	String	The ID of a work assignment that's routed to an agent.

Name	Type	Description
`isEngaged`	Boolean	Indicates whether an agent is working on a work item that's been assigned to them (`true`) or not (`false`).

`getServicePresenceStatusChannels` for Lightning Experience

Retrieves the service channels that are associated with an Omni-Channel user's current presence status.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Get Status Channels" onclick="{! c.getStatusChannels }" />
</aura:component>
```

Controller code:

```
({
    getStatusChannels: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.getServicePresenceStatusChannels().then(function(result) {
            var channels = JSON.parse(result.channels);
            //For example purposes, just retrieve the first channel
            console.log('First channel ID is: ' + channels[0].channelId);
            console.log('First channel developer name is: ' + channels[0].developerName);

        }).catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to an array of `channel` objects, containing the following fields.

Name	Type	Description
`channelId`	String	The ID of the channel.
`developerName`	String	The name of the channel.

`getServicePresenceStatusId` for Lightning Experience

Retrieves an agent's current presence status.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Get Status" onclick="{! c.getStatus }" />
</aura:component>
```

Controller code:

```
({
    getStatus: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.getServicePresenceStatusId().then(function(result) {
            console.log('Status Id is: ' + result.statusId);
        }).catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to an object, containing the following fields.

Name	Type	Description
`statusName`	string	The name of the agent's current presence status.
`statusApiName`	string	The API name of the agent's current presence status.
`statusId`	string	The ID of the agent's current presence status.

`login` for Lightning Experience

Logs an agent in to Omni-Channel with a specific presence status.

Arguments

Name	Type	Description
`statusId`	string	The ID of the presence status. Agents must be given access to this presence status through their associated profile or permission set.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Login" onclick="{! c.login }" />
</aura:component>
```

Controller code:

```
({
    login: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.login({statusId: "0N5xx0000000001"}).then(function(result) {
            if (result) {
                console.log("Login successful");
            } else {
                console.log("Login failed");
            }
        }).catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true` and is rejected on error.

`logout` for Lightning Experience

Logs an agent out of Omni-Channel.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Logout" onclick="{! c.logout }" />
</aura:component>
```

Controller code:

```
({
    logout: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
        omniAPI.logout().then(function(result) {
            if (result) {
                console.log("Logout successful");
            } else {
                console.log("Logout failed");
            }
        }).catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to `true` and is rejected on error.

`lowerAgentWorkFlag` for Lightning Experience

Lowers a flag for this agent work item.

Arguments

Name	Type	Description
`workId`	string	The ID of the work item to lower the flag on.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Lower Flag" onclick="{! c.lowerFlag }" />
</aura:component>
```

Controller code:

```
({
lowerFlag: function(cmp, evt, hlp) {
    var omniAPI = cmp.find("omniToolkit");
    omniAPI.getAgentWorks().then(function(result) {
        var works = JSON.parse(result.works);
        var work = works[0];
        omniAPI.lowerAgentWorkFlag({workId: work.workId}).then(function(res) {
            if (res) {
                console.log("Flag lowered successfully");
            } else {
                console.log("Flag lower failed");
            }
        }).catch(function(error) {
            console.log(error);
        });
    });
}
})
```

Response

This method returns a promise that, upon success, resolves to `true` and is rejected on error.

Note: When Omni-Channel Fallback Mode is enabled, the raise and lower flag feature isn't supported. If this method is used, it doesn't perform any operations and always resolves to true.

SEE ALSO:

Knowledge Article: Routing Work with Omni-Channel Fallback Mode

`raiseAgentWorkFlag` for Lightning Experience

Raises a flag for this agent work item.

Arguments

Name	Type	Description
`workId`	string	The ID of the work item to raise the flag on.
`message`	string	Optional. The message associated with this flag.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Raise Flag" onclick="{! c.raiseFlag }" />
</aura:component>
```

Controller code:

```
({
raiseFlag: function(cmp, evt, hlp) {
    var omniAPI = cmp.find("omniToolkit");
    omniAPI.getAgentWorks().then(function(result) {
        var works = JSON.parse(result.works);
        var work = works[0];
        omniAPI.raiseAgentWorkFlag({workId: work.workId, message: "Raise Flag
Message"}).then(function(res) {
            if (res) {
                console.log("Flag raised successfully");
            } else {
                console.log("Flag raise failed");
            }
        }).catch(function(error) {
            console.log(error);
        });
    });
}
})
```

Response

This method returns a promise that, upon success, resolves to `true` and is rejected on error.

Note: When Omni-Channel Fallback Mode is enabled, the raise and lower flag feature isn't supported. If this method is used, it doesn't perform any operations and always resolves to true.

SEE ALSO:

Knowledge Article: Routing Work with Omni-Channel Fallback Mode

`setServicePresenceStatus` for Lightning Experience

Sets an agent's presence status to a status with a particular ID. If the specified agent is not already logged in, we log in the agent with the presence status. This method removes the need for you to make more calls.

Arguments

Name	Type	Description
`statusId`	string	The ID of the presence status to which you want to set the agent. Agents must be given access to this presence status through their associated profile or permission set.

Sample Code

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <lightning:button label="Set Status" onclick="{! c.setStatus }" />
</aura:component>
```

Controller code:

```
({
    setStatus: function(cmp, evt, hlp) {
        var omniAPI = cmp.find("omniToolkit");
       omniAPI.setServicePresenceStatus({statusId: "0N5xx0000000002"}).then(function(result)
 {
            console.log('Current statusId is: ' + result.statusId);
            console.log('Channel list attached to this status is: ' + result.channels);
        }).catch(function(error) {
            console.log(error);
        });
    }
})
```

Response

This method returns a promise that, upon success, resolves to an object containing the following fields.

Name	Type	Description
`statusName`	string	The name of the agent's current presence status.
`statusApiName`	string	The API name of the agent's current presence status.
`statusId`	string	The ID of the agent's current presence status.
`channels`	JSON string of `channel` objects	Returns the IDs and API names of the channels associated with the presence status.

Events for Lightning Console JavaScript API

Use events and handlers in your Aura components and controllers to respond to events like workspace tabs opening, closing, or gaining focus. In Lightning web components, subscribe to Aura application events using their corresponding Lightning message channels.

Aura application events are received by all rendered Aura components, even those on background tabs. By default, the Lightning message channel events are received by active components only, such as those on the foreground tab or in activated utility bar components. Your component can receive all Lightning message channel events by subscribing with the optional `APPLICATION_SCOPE` parameter. For more information, see Define the Scope of the Message Service.

IN THIS SECTION:

`lightning:tabClosed`

Indicates that a tab has been closed.

Response

Name	Type	Description
`tabId`	string	The ID of the closed tab.

Example: This example prints a line to the browser's developer console when a tab is closed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <aura:handler event="lightning:tabClosed" action="{! c.onTabClosed }"/>
</aura:component>
```

Controller code:

```
({
    onTabClosed : function(component, event, helper) {
        var tabId = event.getParam('tabId');
        console.log("Tab closed: " + tabId);
    }
})
```

`lightning:tabCreated`

Indicates that a tab has been created successfully.

Response

Name	Type	Description
`tabId`	string	The ID of the new tab.

Example: This example prints a line to the browser's developer console when a tab is created, and sets the label of the tab to "New Tab" using the `setTabLabel()` method.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <aura:handler event="lightning:tabCreated" action="{! c.onTabCreated }"/>
</aura:component>
```

Controller code:

```
({
    onTabCreated : function(component, event, helper) {
        console.log("Tab created.");
        var newTabId = event.getParam('tabId');
        var workspaceAPI = component.find("workspace");
        workspaceAPI.setTabLabel({
```

```
            tabId: newTabId,
            label: 'New Tab'
        });
    },
})
```

lightning:tabFocused

Indicates a tab was focused.

`lightning:tabFocused` fires whenever a user selects a workspace tab or subtab, so console navigation users frequently trigger this application event in typical use. This event also fires when going from a tab to a navigation item, or going from a navigation item to a tab. Aura application events notify all listeners registered in the default phase, including listeners in background tabs. Multiple listeners responding at the same time can impact performance. To minimize performance impact, use a utility item as the only listener, or use a custom component event instead.

Response

Name	Type	Description
`previousTabId`	string	The ID of the previously focused tab.
`currentTabId`	string	The ID of the currently focused tab.

Example: This example prints a line to the browser's developer console when a tab is focused, and then returns that tab's `tabInfo` object using the `getTabInfo()` method.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <aura:handler event="lightning:tabFocused" action="{! c.onTabFocused }"/>
</aura:component>
```

Controller code:

```
({
    onTabFocused : function(component, event, helper) {
        console.log("Tab Focused");
        var focusedTabId = event.getParam('currentTabId');
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getTabInfo({
            tabId : focusedTabId
        }).then(function(response) {
            console.log(response);
        });
    }
})
```

lightning:tabRefreshed

Indicates that a tab has been refreshed.

Response

Name	Type	Description
tabId	string	The ID of the refreshed tab.

Example: This example prints a line to the browser's developer console when a tab is refreshed, and then returns that tab's `tabInfo` object using the `getTabInfo()` method.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <aura:handler event="lightning:tabRefreshed" action="{! c.onTabRefreshed }"/>
</aura:component>
```

Controller code:

```
({
       onTabRefreshed : function(component, event, helper) {
           console.log("Tab Refreshed");
           var refreshedTabId = event.getParam("tabId");
           var workspaceAPI = component.find("workspace");
           workspaceAPI.getTabInfo({
                tabId : refreshedTabId
           }).then(function(response) {
                console.log(response);
           });
       }
   })
```

lightning:tabReplaced

Indicates that a tab has been replaced successfully.

Response

Name	Type	Description
tabId	string	The ID of the replaced tab.

Example: This example prints a line to the browser's developer console when a tab is replaced, and then returns that tab's URL using the `getTabURL()` method.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <aura:handler event="lightning:tabReplaced" action="{! c.onTabReplaced }"/>
</aura:component>
```

Controller code:

```
({
    onTabReplaced : function(component, event, helper) {
        console.log("Tab Replaced");
        var replacedTabId = event.getParam("tabId");
        var workspaceAPI = component.find("workspace");
        workspaceAPI.getTabURL({
            tabId : replacedTabId
        }).then(function(response) {
      console.log(response);
        });
    }
})
```

`lightning:tabUpdated`

Indicates that a tab has been updated successfully.

Response

Name	Type	Description
`tabId`	string	The ID of the updated tab.

Example: This example prints a line to the browser's developer console when a tab is updated, and then prints that tab's `tabId`.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:workspaceAPI aura:id="workspace" />
    <aura:handler event="lightning:tabUpdated" action="{! c.onTabUpdated }"/>
</aura:component>
```

Controller code:

```
({
    onTabUpdated : function(component, event, helper) {
        console.log("Tab Updated");
        var updatedTabId = event.getParam("tabId");
        console.log(updatedTabId);
    },
})
```

Subscribe to Aura Application Events in LWC

Subscribe to Aura application events using their corresponding Lightning message channels.

Aura application events are received by all rendered Aura components, even those on background tabs. By default, the Lightning message channel events are received by active components only, such as those on the foreground tab or in activated utility bar components. Your component can receive all Lightning message channel events by subscribing with the optional `APPLICATION_SCOPE` parameter. For more information, see Define the Scope of the Message Service.

IN THIS SECTION:

`lightning__tabClosed`

A Lightning message channel that corresponds to the `lightning:tabClosed` Aura app event.

Response

The response is the same as that of the lightning:tabClosed Aura app event.

LWC Example

Import the `lightning__tabClosed` message channel from the `@salesforce/messageChannel/` scoped module. The event returns the `message` in the response.

```
import { LightningElement, wire } from "lwc";
import { MessageContext, subscribe, unsubscribe, APPLICATION_SCOPE } from
"lightning/messageService";
import tabClosedChannel from "@salesforce/messageChannel/lightning__tabClosed";

export default class TabClosedExample extends LightningElement {
    subscription = null;
```

```
    @wire(MessageContext) messageContext;

    // Encapsulate logic for Lightning message service subscribe and unsubscribe
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                tabClosedChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    unsubscribeToMessageChannel() {
        unsubscribe(this.subscription);
        this.subscription = null;
    }

    // Handler for message received by component
    handleMessage(message) {
        // do something
        console.log("Tab closed: ", message.tabId);
    }
    // Standard lifecycle hooks used to subscribe and unsubscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    disconnectedCallback() {
        this.unsubscribeToMessageChannel();
    }
```

For more information, see Subscribe and Unsubscribe from a Message Channel.

`lightning__tabCreated`

A Lightning message channel that corresponds to the `lightning:tabCreated` Aura app event. This message channel is available for Lightning web components used within a Lightning console app.

Response

The response is the same as that of the lightning:tabCreated Aura app event.

LWC Example

Import the `lightning__tabCreated` message channel from the `@salesforce/messageChannel/` scoped module. The event returns the `message` in the response.

```
import { LightningElement, track, wire } from "lwc";
import { MessageContext, subscribe, unsubscribe, APPLICATION_SCOPE } from
"lightning/messageService";
import tabCreatedChannel from "@salesforce/messageChannel/lightning__tabCreated";
```

```
export default class TabCreatedExample extends LightningElement {
    subscription = null;
    @wire(MessageContext) messageContext;

    // Encapsulate logic for Lightning message service subscribe and unsubscribe
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                tabCreatedChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    unsubscribeToMessageChannel() {
        unsubscribe(this.subscription);
        this.subscription = null;
    }

    // Handler for message received by component
    handleMessage(message) {
        // do something
    }
    // Standard lifecycle hooks used to subscribe and unsubscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    disconnectedCallback() {
        this.unsubscribeToMessageChannel();
    }
```

For more information, see Subscribe and Unsubscribe from a Message Channel.

`lightning__tabFocused`

A Lightning message channel that corresponds to the `lightning:tabFocused` Aura app event. This message channel is available for Lightning web components used within a Lightning console app.

By default, this event is only received when that component's tab comes into focus, not when it leaves focus. To receive all events and minimize performance impact, use a utility item as the only listener.

Response

The response is the same as that of the lightning:tabFocused Aura app event.

LWC Example

Import the `lightning__tabFocused` message channel from the `@salesforce/messageChannel/` scoped module. The event returns the `message` in the response.

```
import { LightningElement, track, wire } from "lwc";
import { MessageContext, subscribe, unsubscribe, APPLICATION_SCOPE } from
"lightning/messageService";
import tabFocusedChannel from "@salesforce/messageChannel/lightning__tabFocused";

export default class TabFocusedExample extends LightningElement {
    subscription = null;
    @wire(MessageContext) messageContext;

    // Encapsulate logic for Lightning message service subscribe and unsubscribe
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                tabFocusedChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    unsubscribeToMessageChannel() {
        unsubscribe(this.subscription);
        this.subscription = null;
    }

    // Handler for message received by component
    handleMessage(message) {
        // do something
    }
    // Standard lifecycle hooks used to subscribe and unsubscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    disconnectedCallback() {
        this.unsubscribeToMessageChannel();
    }
```

For more information, see Subscribe and Unsubscribe from a Message Channel.

`lightning__tabRefreshed`

A Lightning message channel that corresponds to the `lightning:tabRefreshed` Aura app event. This message channel is available for Lightning web components used within a Lightning console app.

Response

The response is the same as that of the lightning:tabRefreshed Aura app event.

LWC Example

Import the `lightning__tabRefreshed` message channel from the `@salesforce/messageChannel/` scoped module. The event returns the `message` in the response.

```
import { LightningElement, track, wire } from "lwc";
import { MessageContext, subscribe, unsubscribe, APPLICATION_SCOPE } from
"lightning/messageService";
import tabRefreshedChannel from "@salesforce/messageChannel/lightning__tabRefreshed";

export default class TabRefreshedExample extends LightningElement {
    subscription = null;
    @wire(MessageContext) messageContext;

    // Encapsulate logic for Lightning message service subscribe and unsubscribe
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                tabRefreshedChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    unsubscribeToMessageChannel() {
        unsubscribe(this.subscription);
        this.subscription = null;
    }

    // Handler for message received by component
    handleMessage(message) {
        // do something
    }
    // Standard lifecycle hooks used to subscribe and unsubscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    disconnectedCallback() {
        this.unsubscribeToMessageChannel();
    }
```

For more information, see Subscribe and Unsubscribe from a Message Channel.

`lightning__tabReplaced`

A Lightning message channel that corresponds to the `lightning:tabReplaced` Aura app event. This message channel is available for Lightning web components used within a Lightning console app.

Response

The response is the same as that of the lightning:tabReplaced Aura app event.

LWC Example

Import the `lightning__tabReplaced` message channel from the `@salesforce/messageChannel/` scoped module. The event returns the `message` in the response.

```
import { LightningElement, wire } from "lwc";
import { MessageContext, subscribe, unsubscribe, APPLICATION_SCOPE } from
"lightning/messageService";
import tabReplacedChannel from "@salesforce/messageChannel/lightning__tabReplaced";

export default class TabReplacedExample extends LightningElement {
    subscription = null;
    @wire(MessageContext) messageContext;

    // Encapsulate logic for Lightning message service subscribe and unsubscribe
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                tabReplacedChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    unsubscribeToMessageChannel() {
        unsubscribe(this.subscription);
        this.subscription = null;
    }

    // Handler for message received by component
    handleMessage(message) {
        // do something
    }
    // Standard lifecycle hooks used to subscribe and unsubscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    disconnectedCallback() {
        this.unsubscribeToMessageChannel();
    }
```

For more information, see Subscribe and Unsubscribe from a Message Channel.

`lightning__tabUpdated`

A Lightning message channel that corresponds to the `lightning:tabUpdated` Aura app event. This message channel is available for Lightning web components used within a Lightning console app.

Response

The response is the same as that of the lightning:tabUpdated Aura app event.

LWC Example

Import the `lightning__tabUpdated` message channel from the `@salesforce/messageChannel/` scoped module. The event returns the `message` in the response.

```
import { LightningElement, track, wire } from "lwc";
import { MessageContext, subscribe, unsubscribe, APPLICATION_SCOPE } from
"lightning/messageService";
import tabUpdatedChannel from "@salesforce/messageChannel/lightning__tabUpdated";

export default class TabUpdatedExample extends LightningElement {
    subscription = null;
    @wire(MessageContext) messageContext;

    // Encapsulate logic for Lightning message service subscribe and unsubscribe
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                tabUpdatedChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    unsubscribeToMessageChannel() {
        unsubscribe(this.subscription);
        this.subscription = null;
    }

    // Handler for message received by component
    handleMessage(message) {
        // do something
    }
    // Standard lifecycle hooks used to subscribe and unsubscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    disconnectedCallback() {
        this.unsubscribeToMessageChannel();
    }
```

For more information, see Subscribe and Unsubscribe from a Message Channel.

Lightning Web Component Events for Enhanced Messaging

JavaScript can be executed when certain types of events occur in a console, such as when a user closes a tab. A few events are specific to Messaging. These events apply to Lightning web components in Lightning Experience only.

Note: Use only rendered components with the Conversation Toolkit APIs. If you use a component that doesn't have markup or that operates in the background, the APIs don't work. The conversation component must also be rendered for the APIs to work.

IN THIS SECTION:

`lightning__conversationAgentSend`

Messaging event triggered when an agent sends a message through the Salesforce console. This method intercepts the message before it's sent to the chat visitor. This event is also triggered when using Enhanced Messaging channels. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current conversation.
`content`	String	The text of the message in the conversation log.
`name`	String	The name of the agent who is attempting to send the message. This name matches the agent name displayed in the conversation log.
`timestamp`	Date/Time	The date and time that the agent attempted to send the message.

LWC Sample Code

To listen to the `lightning__conversationAgentSend` event, import the Lightning Message Service features from `lightning/messageService` and pass the event to the `subscribe()` method.

```
import { LightningElement, wire } from 'lwc';

import {
    subscribe,
    unsubscribe,
    APPLICATION_SCOPE,
```

```
    MessageContext
} from 'lightning/messageService';

import ConversationAgentSendChannel from
'@salesforce/messageChannel/lightning__conversationAgentSend';

export default class ConversationAgentSendExample extends LightningElement {
    subscription = null;
    recordId;

    // To pass scope, you must get a message context.
    @wire(MessageContext)
    messageContext;

    // Standard lifecycle hook used to subscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    // Pass scope to the subscribe() method.
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                ConversationAgentSendChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    // Handler for message received by component
    handleMessage(message) {
        this.recordId = message.recordId;
    }
}
```

Aura Components Sample Code

Component code:

```
<lightning:messageChannel type="lightning__conversationAgentSend" scope="APPLICATION"
                          onMessage="{!c.onConversationAgentSend}" />
```

Controller code:

```
({
    onConversationAgentSend: function(cmp, evt, helper) {
        var recordId = evt.getParam("recordId");
        var content = evt.getParam("content");
        var name = evt.getParam("name");
        var timestamp = evt.getParam("timestamp");

        console.log("recordId:" + recordId + " content:" + content + " name:" + name + "
timestamp:" + timestamp);
```

```
    }
})
```

`lightning__conversationEnded`

Messaging event triggered when an active chat ends or an agent leaves a chat conference. This event is also triggered when using Enhanced Messaging channels. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current chat.

LWC Sample Code

To listen to the `lightning__conversationEnded` event, import the Lightning Message Service features from `lightning/messageService` and pass the event to the `subscribe()` method.

```
import { LightningElement, wire } from 'lwc';

import {
    subscribe,
    unsubscribe,
    APPLICATION_SCOPE,
    MessageContext
} from 'lightning/messageService';

import ConversationEndedChannel from
'@salesforce/messageChannel/lightning__conversationEnded';

export default class ConversationEndedExample extends LightningElement {
    subscription = null;
    recordId;

    // To pass scope, you must get a message context.
    @wire(MessageContext)
    messageContext;

    // Standard lifecycle hook used to subscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    // Pass scope to the subscribe() method.
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
```

```
                ConversationEndedChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    // Handler for message received by component
    handleMessage(message) {
        this.recordId = message.recordId;
    }
}
```

Aura Components Sample Code

Component code:

```
<lightning:messageChannel type="lightning__conversationEnded" scope="APPLICATION"
                          onMessage="{!c.onConversationEndedEvent}" />
```

Controller code:

```
({
    onConversationEndedEvent: function(cmp, evt, helper) {
        var conversation = cmp.find("conversationKit");
        var recordId = evt.getParam("recordId");

        console.log("recordId:" + recordId);
    }
})
```

lightning__conversationEndUserMessage

Messaging event triggered when the customer sends a new message. In Enhanced Messaging channels, this event is triggered only for text messages. This event is not triggered for messages with files or rich content. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

Response

Name	Type	Description
recordId	String	The ID of the work record that's associated with the current conversation.
content	String	The message sent by the customer.
name	String	The name of the user who sent the message. This name matches the username displayed in the conversation log.
timestamp	Date/Time	The date and time the message was received.

LWC Sample Code

To listen to the `lightning__conversationEndUserMessage` event, import the Lightning Message Service features from `lightning/messageService` and pass the event to the `subscribe()` method.

```
import { LightningElement, wire } from 'lwc';

import {
    subscribe,
    unsubscribe,
    APPLICATION_SCOPE,
    MessageContext
} from 'lightning/messageService';

import ConversationEndUserChannel from
'@salesforce/messageChannel/lightning__conversationEndUserMessage';

export default class ConversationEndUserExample extends LightningElement {
    subscription = null;
    recordId;

    // To pass scope, you must get a message context.
    @wire(MessageContext)
    messageContext;

    // Standard lifecycle hook used to subscribe to the message channel
    connectedCallback() {
        this.subscribeToMessageChannel();
    }

    // Pass scope to the subscribe() method.
    subscribeToMessageChannel() {
        if (!this.subscription) {
            this.subscription = subscribe(
                this.messageContext,
                ConversationEndUserChannel,
                (message) => this.handleMessage(message),
                { scope: APPLICATION_SCOPE }
            );
        }
    }

    // Handler for message received by component
    handleMessage(message) {
        this.recordId = message.recordId;
    }
}
```

Aura Components Sample Code

Component code:

```
<lightning:messageChannel type="lightning__conversationEndUserMessage" scope="APPLICATION"

                          onMessage="{!c.onConversationEndUserMessage}" />
```

Controller code:

```
({
    onConversationEndUserMessage: function(cmp, evt, helper) {
        var recordId = evt.getParam('recordId');
        var content = evt.getParam('content');
        var name = evt.getParam('name');
        var timestamp = evt.getParam('timestamp');

        console.log("recordId:" + recordId + " content:" + content + " name:" + name + "
timestamp:" + timestamp);
    }
})
```

Aura Events for Enhanced Messaging

JavaScript can be executed when certain types of events occur in a console, such as when a user closes a tab. A few events are specific to Messaging. These events apply to Aura components in Lightning Experience only.

IN THIS SECTION:

`lightning:conversationAgentSend`

Messaging event triggered when an agent sends a message through the Salesforce console. This method intercepts the message before it's sent to the chat visitor. This event is also triggered when using Enhanced Messaging channels. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current conversation.
`content`	String	The text of the message in the conversation log.

Name	Type	Description
`name`	String	The name of the agent who is attempting to send the message. This name matches the agent name displayed in the conversation log.
`type`	String	The type of message that was received, such as an Agent or EndUser message.
`timestamp`	Date/Time	The date and time that the agent attempted to send the message.

Example: Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <aura:handler event="lightning:conversationAgentSend" action="{! c.onAgentSend}" />
</aura:component>
```

Controller code:

```
({
    onAgentSend: function(cmp, evt, helper) {
        var recordId = evt.getParam("recordId");
        var content = evt.getParam("content");
        var name = evt.getParam("name");
        var type = evt.getParam("type");
        var timestamp = evt.getParam("timestamp");

        console.log("recordId:" + recordId + " content:" + content + " name:" + name
+ " timestamp:" + timestamp);
    }
})
```

`lightning:conversationChatEnded`

Messaging event triggered when an active chat ends or an agent leaves a chat conference. This event is also triggered when using Enhanced Messaging channels. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current chat.

Example: Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <aura:handler event="lightning:conversationChatEnded" action="{!c.onChatEnded}" />
</aura:component>
```

Controller code:

```
({
    onChatEnded: function(cmp, evt, helper) {
        var conversation = cmp.find("conversationKit");
        var recordId = evt.getParam("recordId");

        console.log("recordId:" + recordId);
    }
})
```

`lightning:conversationNewMessage`

Messaging event triggered when the customer sends a new message. In Enhanced Messaging channels, this event is triggered only for text messages. This event is not triggered for messages with files or rich content. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current conversation.
`content`	String	The message sent by the customer.
`name`	String	The name of the user who sent the message. This name matches the username displayed in the conversation log.
`type`	String	The type of message that was received, such as an Agent or Visitor message.
`timestamp`	Date/Time	The date and time the message was received.

Example: Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <aura:handler event="lightning:conversationNewMessage" action="{!c.onNewMessage}"
/>
</aura:component>
```

Controller code:

```
({
    onNewMessage: function(cmp, evt, helper) {
        var recordId = evt.getParam('recordId');
        var content = evt.getParam('content');
        var name = evt.getParam('name');
        var type = evt.getParam('type');
        var timestamp = evt.getParam('timestamp');

        console.log("recordId:" + recordId + " content:" + content + " name:" + name
+ " timestamp:" + timestamp);
    }
})
```

Events for Chat

JavaScript can be executed when certain types of events occur in a console, such as when a user closes a tab. A few events are specific to Chat. These events apply to Lightning Experience only.

Important: The legacy chat product is in maintenance-only mode, and we won't continue to build new features. You can continue to use it, but we no longer recommend that you implement new chat channels. Instead, you can modernize your customer communication with Messaging for In-App and Web. Messaging offers many of the chat features that you love plus asynchronous conversations that can be picked back up at any time. For Lightning Console JavaScript API, use Lightning Web Component Events for Enhanced Messaging or Aura Events for Enhanced Messaging.

IN THIS SECTION:

lightning:conversationAgentSend
Event triggered when an agent sends a chat message through the Salesforce console. This method does not intercept the message before it's sent to the chat visitor. This event is also triggered when using Enhanced Messaging channels.

lightning:conversationChatEnded
Event triggered when an active chat ends or an agent leaves a chat conference. This event is also triggered when using Enhanced Messaging channels.

lightning:conversationCustomEvent
Event triggered when a custom event occurs during a chat.

lightning:conversationNewMessage
Event triggered when the customer sends a new message. This event is also triggered when using Enhanced Messaging channels. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

`lightning:conversationAgentSend`

Event triggered when an agent sends a chat message through the Salesforce console. This method does not intercept the message before it's sent to the chat visitor. This event is also triggered when using Enhanced Messaging channels.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current chat.
`content`	String	The text of the message in the chat log.
`name`	String	The name of the agent who is attempting to send the message. This name matches the agent name displayed in the chat log.
`type`	String	The type of message that was received—for example, agent.
`timestamp`	Date/Time	The date and time that the agent attempted to send the chat message.

Example: Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <aura:handler event="lightning:conversationAgentSend" action="{! c.onAgentSend}" />
</aura:component>
```

Controller code:

```
({
    onAgentSend: function(cmp, evt, helper) {
        var recordId = evt.getParam("recordId");
        var content = evt.getParam("content");
        var name = evt.getParam("name");
        var type = evt.getParam("type");
        var timestamp = evt.getParam("timestamp");

        console.log("recordId:" + recordId + " content:" + content + " name:" + name
+ " timestamp:" + timestamp);
    }
})
```

`lightning:conversationChatEnded`

Event triggered when an active chat ends or an agent leaves a chat conference. This event is also triggered when using Enhanced Messaging channels.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current chat.

Example: Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <aura:handler event="lightning:conversationChatEnded" action="{!c.onChatEnded}" />
</aura:component>
```

Controller code:

```
({
    onChatEnded: function(cmp, evt, helper) {
        var conversation = cmp.find("conversationKit");
        var recordId = evt.getParam("recordId");

        console.log("recordId:" + recordId);
    }
})
```

`lightning:conversationCustomEvent`

Event triggered when a custom event occurs during a chat.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current chat.
`type`	String	The type of the custom event that was sent to this chat; corresponds to the `type` argument of the `liveagent.chasitor.sendCustomEvent()` method used to send this event from the chat window.
`data`	String	The data of the custom event that was sent to this chat; corresponds to the `data` argument of the `liveagent.chasitor.sendCustomEvent()` method used to send this event from the chat window.

Example: Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
  <aura:handler event="lightning:conversationCustomEvent" action="{!c.onCustomEvent}"
 />
</aura:component>
```

Controller code:

```
({
    onCustomEvent: function(cmp, evt, helper) {
        var conversation = cmp.find("conversationKit");
        var data = evt.getParam("data");
        var type = evt.getParam("type");

        console.log("type:" + type + " data:" + data);
    }
})
```

`lightning:conversationNewMessage`

Event triggered when the customer sends a new message. This event is also triggered when using Enhanced Messaging channels. To work with Enhanced Messaging channels, the session must be active and the Enhanced Conversation Component must be visible on the page.

Response

Name	Type	Description
`recordId`	String	The ID of the work record that's associated with the current chat.
`content`	String	The text sent by the customer.
`name`	String	The name of the user who sent the message. This name matches the username displayed in the chat log.
`type`	String	The type of message that was received, such as an Agent or Visitor message.
`timestamp`	Date/Time	The date and time the message was received.

Example: Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global"
description="Conversation toolkit api sample">
  <lightning:conversationToolkitAPI aura:id="conversationKit" />
```

```
  <aura:handler event="lightning:conversationNewMessage" action="{!c.onNewMessage}"
/>
</aura:component>
```

Controller code:

```
({
    onNewMessage: function(cmp, evt, helper) {
        var recordId = evt.getParam('recordId');
        var content = evt.getParam('content');
        var name = evt.getParam('name');
        var type = evt.getParam('type');
        var timestamp = evt.getParam('timestamp');

        console.log("recordId:" + recordId + " content:" + content + " name:" + name
+ " timestamp:" + timestamp);
    }
})
```

Events for Omni-Channel

JavaScript can be executed when certain types of events occur in a console, such as when a user closes a tab. There are a few events that are specific to Omni-Channel. These events apply to Lightning Experience only.

IN THIS SECTION:

lightning:omniChannelConnectionError
Indicates that a network connection issue occurred.

lightning:omniChannelLoginSuccess
Indicates that an agent has been logged into Omni-Channel successfully.

lightning:omniChannelStatusChanged
Indicates that an agent has changed his or her presence status in Omni-Channel.

lightning:omniChannelLogout
Indicates that an agent has logged out of Omni-Channel.

lightning:omniChannelWorkAssigned
Indicates that an agent has been assigned a new work item.

lightning:omniChannelWorkAccepted
Indicates that an agent has accepted a work assignment, or that a work assignment has been automatically accepted.

lightning:omniChannelWorkDeclined
Indicates that an agent has declined a work assignment.

lightning:omniChannelWorkClosed
Indicates that the status of an AgentWork object is changed to Closed.

lightning:omniChannelWorkFlagUpdated
Indicates that an agent's work item flag has been raised or lowered.

lightning:omniChannelWorkloadChanged

Indicates that an agent's workload has changed. This includes receiving new work items, declining work items, and closing items in the console. It also indicates that there has been a change to an agent's capacity or presence configuration, or that the agent has gone offline in the Omni-Channel utility.

`lightning:omniChannelConnectionError`

Indicates that a network connection issue occurred.

Response

Name	Type	Description
`error`	object	The network connection error message.

Example: This example prints a line to the browser's developer console when a network connection error occurs.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <aura:handler event="lightning:omniChannelConnectionError" action="{!
c.onConnectionError }"/>
</aura:component>
```

Controller code:

```
({
    onConnectionError : function(component, event, helper) {
        console.log("Network Connection Error.");
        var error = event.getParam('error');
        console.log(error);
    },
})
```

`lightning:omniChannelLoginSuccess`

Indicates that an agent has been logged into Omni-Channel successfully.

Response

Name	Type	Description
`statusId`	string	The ID of the agent's current presence status.

Example: This example prints a line to the browser's developer console when an Omni-Channel user logs into Omni-Channel successfully.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
   <aura:handler event="lightning:omniChannelLoginSuccess" action="{! c.onLoginSuccess
 }"/>
</aura:component>
```

Controller code:

```
({
    onLoginSuccess : function(component, event, helper) {
        console.log("Login success.");
        var statusId = event.getParam('statusId');
        console.log(statusId);
    },
})
```

`lightning:omniChannelStatusChanged`

Indicates that an agent has changed his or her presence status in Omni-Channel.

Response

Name	Type	Description
`statusId`	string	The ID of the agent's current presence status.
`channels`	string	JSON string of channel objects.
`statusName`	string	The name of the agent's current presence status.
`statusApiName`	string	The API name of the agent's current presence status.

Example: This example prints status details to the browser's developer console when an Omni-Channel user's presence status is changed.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
   <aura:handler event="lightning:omniChannelStatusChanged" action="{! c.onStatusChanged
 }"/>
</aura:component>
```

Controller code:

```
({
    onStatusChanged : function(component, event, helper) {
        console.log("Status changed.");
        var statusId = event.getParam('statusId');
        var channels = event.getParam('channels');
        var statusName = event.getParam('statusName');
        var statusApiName = event.getParam('statusApiName');
        console.log(statusId);
        console.log(channels);
        console.log(statusName);
        console.log(statusApiName);
    },
})
```

`channel`

The `channel` object contains the following properties:

Name	Type	Description
`channelId`	string	Retrieves the ID of the service channel that's associated with a presence status.
`developerName`	string	Retrieves the developer name of the service channel that's associated with the `channelId`.

`lightning:omniChannelLogout`

Indicates that an agent has logged out of Omni-Channel.

Response

Name	Type	Description
`reason`	string	The reason why the agent is logged out. Possible values are: • DuplicateLogin • DuplicateLoginInSameBrowser • MaintenanceLogout • OmniSupervisorLogout

Example: This example prints a line to the browser's developer console when an Omni-Channel user logs out of Omni-Channel.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <aura:handler event="lightning:omniChannelLogout" action="{! c.onLogout }"/>
</aura:component>
```

Controller code:

```
({
    onLogout : function(component, event, helper) {
        console.log("Logout success.");
        var reason = event.getParam('reason');
        console.log(reason);
    },
})
```

`lightning:omniChannelWorkAssigned`

Indicates that an agent has been assigned a new work item.

Response

Name	Type	Description
`workItemId`	string	The ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent.
`workId`	string	The ID of a work assignment that's routed to an agent.

Example: This example prints work details to the browser's developer console when an Omni-Channel user is assigned a new work item.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
   <aura:handler event="lightning:omniChannelWorkAssigned" action="{! c.onWorkAssigned
 }"/>
</aura:component>
```

Controller code:

```
({
    onWorkAssigned : function(component, event, helper) {
        console.log("Work assigned.");
        var workItemId = event.getParam('workItemId');
        var workId = event.getParam('workId');
```

```
        console.log(workItemId);
        console.log(workId);
    },
})
```

lightning:omniChannelWorkAccepted

Indicates that an agent has accepted a work assignment, or that a work assignment has been automatically accepted.

Response

Name	Type	Description
workItemId	string	The ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a workId when it's assigned to an agent.
workId	string	The ID of a work assignment that's routed to an agent.

Example: This example prints work details to the browser's developer console when an Omni-Channel user accepts a work assignment, or when a work assignment is automatically accepted.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
   <aura:handler event="lightning:omniChannelWorkAccepted" action="{! c.onWorkAccepted
 }"/>
</aura:component>
```

Controller code:

```
({
    onWorkAccepted : function(component, event, helper) {
        console.log("Work accepted.");
        var workItemId = event.getParam('workItemId');
        var workId = event.getParam('workId');
        console.log(workItemId);
        console.log(workId);
    },
})
```

lightning:omniChannelWorkDeclined

Indicates that an agent has declined a work assignment.

Response

Name	Type	Description
`workItemId`	string	The ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent.
`workId`	string	The ID of a work assignment that's routed to an agent.

Example: This example prints work details to the browser's developer console when an Omni-Channel user declines a work assignment.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
   <aura:handler event="lightning:omniChannelWorkDeclined" action="{! c.onWorkDeclined
 }"/>
</aura:component>
```

Controller code:

```
({
    onWorkDeclined : function(component, event, helper) {
        console.log("Work declined.");
        var workItemId = event.getParam('workItemId');
        var workId = event.getParam('workId');
        console.log(workItemId);
        console.log(workId);
    },
})
```

`lightning:omniChannelWorkClosed`

Indicates that the status of an AgentWork object is changed to Closed.

Response

Name	Type	Description
`workItemId`	string	The ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent.
`workId`	string	The ID of a work assignment that's routed to an agent.

Example: This example prints work details to the browser's developer console when an Omni-Channel user closes a tab in the console that's associated with a work item.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <aura:handler event="lightning:omniChannelWorkClosed" action="{! c.onWorkClosed
}"/>
</aura:component>
```

Controller code:

```
({
    onWorkClosed : function(component, event, helper) {
        console.log("Work closed.");
        var workItemId = event.getParam('workItemId');
        var workId = event.getParam('workId');
        console.log(workItemId);
        console.log(workId);
    },
})
```

`lightning:omniChannelWorkFlagUpdated`

Indicates that an agent's work item flag has been raised or lowered.

Response

Name	Type	Description
`workId`	string	The ID of a work item with the updated flag.
`isFlagged`	Boolean	Specifies whether the flag is raised or not.
`message`	string	Optional. A message associated with changing the flag.
`roleUpdatedBy`	string	The role of the user who triggered this flag change. The value is `AGENT` or `SUPERVISOR`.
`updatedBy`	string	The ID of the user who triggered this flag change.

Example: This example prints a line to the browser's developer console when an agent's work item flag is raised or lowered.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
    <aura:handler event="lightning:omniChannelWorkFlagUpdated" action="{!
```

```
c.onChannelWorkFlagUpdated }"/>
</aura:component>
```

Controller code:

```
({
onChannelWorkFlagUpdated : function(cmp, evt, hlp) {
    var workId = evt.getParam('workId');
    var message = evt.getParam('message');
    var isFlagged = evt.getParam('isFlagged');
    console.log("WorkFlag  event");
    console.log("  workId    : "+ workId);
    console.log("  isFlagged : "+ isFlagged);
    console.log("  message   : "+ message);
}
})
```

`lightning:omniChannelWorkloadChanged`

Indicates that an agent's workload has changed. This includes receiving new work items, declining work items, and closing items in the console. It also indicates that there has been a change to an agent's capacity or presence configuration, or that the agent has gone offline in the Omni-Channel utility.

Response

Name	Type	Description
`configuredCapacity`	number	The configured primary capacity for the agent.
`previousWorkload`	number	The agent's primary workload before the change.
`newWorkload`	number	The agent's new primary workload after the change.
`configuredInterruptibleCapacity`	number	The configured interruptible capacity for the agent.
`previousInterruptibleWorkload`	number	The agent's interruptible workload before the change.
`newInterruptibleWorkload`	number	The agent's new interruptible workload after the change.

Example: This example prints workload details to the browser's developer console when an agent's workload changes.

Component code:

```
<aura:component implements="flexipage:availableForAllPageTypes" access="global" >
    <lightning:omniToolkitAPI aura:id="omniToolkit" />
```

```
    <aura:handler event="lightning:omniChannelWorkloadChanged" action="{!
c.onWorkloadChanged }"/>
</aura:component>
```

Controller code:

```
({
    onWorkloadChanged : function(component, event, helper) {
        console.log("Workload changed.");
        var configuredCapacity = event.getParam('configuredCapacity');
        var previousWorkload = event.getParam('previousWorkload');
        var newWorkload = event.getParam('newWorkload');
        console.log(configuredCapacity);
        console.log(previousWorkload);
        console.log(newWorkload);
    },
})
```

CHAPTER 3 Salesforce Console Integration Toolkit for Salesforce Classic

The Salesforce Console Integration Toolkit is a browser-based JavaScript API that provides you with programmatic access to the console in Salesforce Classic. The Salesforce Console Integration Toolkit uses browsers as clients to display pages as tabs in the console. For example, the toolkit lets you integrate third-party systems with the console, opening up an external application in the same window, in a tab.

This guide explains how to use the Salesforce Console Integration Toolkit in JavaScript to embed API calls and processes. The toolkit is available for use with third-party domains, such as `www.yourdomain.com`; however, the examples in this guide are in Visualforce. The functionality it describes is available to your organization if you have:

- Enterprise, Unlimited, Performance, or Developer Edition with the Service Cloud
- Salesforce console

The Salesforce Console Integration Toolkit supports any browser that the Salesforce console supports.

Note: To enable the toolkit for third-party domains, add the domains to the allowlist of the Salesforce console.

EDITIONS

Available in: Salesforce Classic (not available in all orgs)

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

IN THIS SECTION:

Methods for Salesforce Classic

If your org is using Salesforce Classic, use Salesforce Console Integration Toolkit methods.

SEE ALSO:

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

Salesforce Help: Supported Browsers and Devices

Methods for Salesforce Classic

When to Use the Salesforce Console Integration Toolkit

The Salesforce Console Integration Toolkit helps advanced administrators and developers implement custom functionality for the Salesforce console. For example, you can use the Salesforce Console Integration Toolkit to display Visualforce pages or third-party content as tabs in the Salesforce console. The Salesforce Console Integration Toolkit is an API that uses browsers as clients to display pages in the console.

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

The following table lists additional features that developers can use to implement custom functionality for Salesforce organizations.

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. Due to third-party cookie restrictions in modern web browsers, Visualforce pages can't load in Salesforce Classic console apps when third-party cookies are disabled. See Visualforce Limitations in Salesforce Classic When Third-Party Cookies are Blocked.
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).

Feature	Description
	• 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.

Salesforce Console Integration Toolkit Support Policy

The current release of the Salesforce Console Integration Toolkit is the only version that receives enhancements.

Previous versions may or may not receive fixes. When a new version is released, the previous version remains available.

IN THIS SECTION:

Backward Compatibility

Salesforce strives to make backward compatibility easy when using the Salesforce Console Integration Toolkit.

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

The Salesforce Console Integration Toolkit matches the API version for any given release. For example, if the current version of SOAP API is 63.0, then there's also a version 63.0 of Salesforce Console Integration Toolkit.

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

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

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

End-of-Life

Salesforce is committed to supporting each Salesforce Console Integration Toolkit version for a minimum of three years from the date of its first release. To improve the quality and performance of the Salesforce Console Integration Toolkit, versions that are more than three years old may not be supported.

When a Salesforce Console Integration Toolkit version is scheduled to be unsupported, an advance end-of-life notice will be given at least one year before support for the version ends. Salesforce will directly notify customers using Salesforce Console Integration Toolkit versions scheduled for end of life.

Change a Visualforce Page by Using the Salesforce Console Integration Toolkit

Each implementation of Salesforce Console Integration Toolkit can look different. This example shows how to change the Salesforce console user interface using the Salesforce Console Integration Toolkit.

Note: Due to third-party cookie restrictions in modern web browsers, Visualforce pages can't load in Salesforce Classic console apps when third-party cookies are disabled. See Visualforce Limitations in Salesforce Classic When Third-Party Cookies are Blocked.

1. Create a Visualforce page.
2. Cut and paste the following sample code into your Visualforce page.

   This code demonstrates various functions of the Salesforce Console Integration Toolkit:

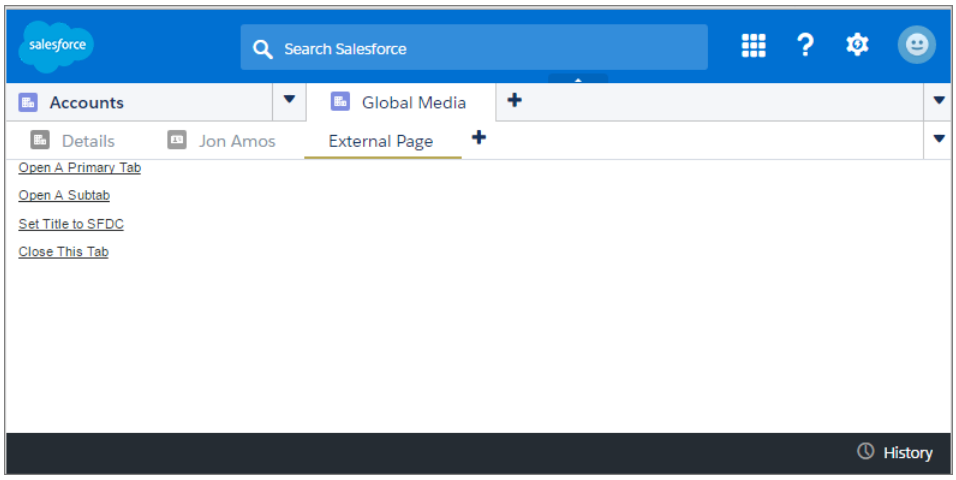
```
<apex:page standardController="Case">

 <apex:includeScript value="/support/console/63.0/integration.js"/>
 <script type="text/javascript">
        function openPrimaryTab() {
            sforce.console.openPrimaryTab(undefined,
               'https://www.example.com', true, 'example');
        }

        //The callback function that openSubtab will call once it has the ID for its
primary tab
        var callOpenSubtab=function callOpenSubtab(result) {
            sforce.console.openSubtab(result.id,
               'https://www.example.com', true, 'example');
        };

        function openSubtab() {
            sforce.console.getEnclosingPrimaryTabId(callOpenSubtab);
        }

        //Sets the title of the current tab to "Example"
        function setTitle() {
            sforce.console.setTabTitle('Example');
        }

        //The callback function that closeTab will call once it has the ID for its tab
        var callCloseTab= function callCloseTab(result) {
            sforce.console.closeTab(result.id);
        }
```

```
        function closeTab() {
            sforce.console.getEnclosingTabId(callCloseTab);
        }
    </script>

     <A HREF="#" onClick="openPrimaryTab();return false">Open A Primary Tab</A>
     <p/><A HREF="#" onClick="openSubtab();return false">Open A Subtab</A>
     <p/><A HREF="#" onClick="setTitle();return false">Set Title to Example</A>
     <p/><A HREF="#" onClick="closeTab();return false">Close This Tab</A>

</apex:page>
```

3. Create a custom link for cases that use your Visualforce page.
4. Edit the page layout for cases and add your custom link.
5. Add any domains to the console's allowlist.

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Here's the sample Visualforce page loaded in the console.



SEE ALSO:

Visualforce Developer Guide

Salesforce Help: Create and Edit Page Layouts

Salesforce Help: Allow Domains for a Salesforce Console in Salesforce

Working with the Salesforce Console Integration Toolkit

You can use Salesforce Console Integration Toolkit to streamline a business process.

With Salesforce Console Integration Toolkit, you can:

- Open a new primary tab or subtab that displays a specified URL
- Set the title of a primary tab or a subtab

- Return the ID of a primary tab or subtab
- Close a specified primary tab or subtab

Before developing an Salesforce Console Integration Toolkit implementation, learn how to connect to Salesforce Console Integration Toolkit and review the best practices.

IN THIS SECTION:

Connect to the Toolkit

The first portion of any JavaScript code that uses the Salesforce Console Integration Toolkit 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 in a third-party domain.

Note: Due to third-party cookie restrictions in modern web browsers, Visualforce pages can't load in Salesforce Classic console apps when third-party cookies are disabled. See Visualforce Limitations in Salesforce Classic When Third-Party Cookies are Blocked.

The version of the Salesforce Console Integration Toolkit is in the URL.

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

```
<apex:page>
          <script src="/support/console/63.0/integration.js"
type="text/javascript"></script>
     ...
</apex:page>
```

For Visualforce, a relative path is sufficient to include `integration.js`, and is recommended.

- For a third-party domain, insert this `<script>` tag.

```
<script
src="https://MyDomainName--PackageName.vf.force.com/support/console/63.0/integration.js"
 type="text/javascript"></script>
```

SEE ALSO:

Salesforce Help: My Domain URL Formats

Asynchronous Calls with the Salesforce Console Integration Toolkit

The Salesforce Console Integration Toolkit 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 additional parameter with the API call, which is referred to as a callback function. Once the result is ready, the server invokes the callback method with the result.

Asynchronous syntax:

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

For example:

```
//Open a new primary tab with the Salesforce home page in it
   sforce.console.openPrimaryTab(null, 'https://salesforce.com',
      false, 'Salesforce', callback);
```

Working with Lightning Platform Canvas

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

Lightning Platform Canvas and the Salesforce Console Integration Toolkit 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 Lightning Platform Canvas, is the ability to choose authentication methods. For more information, see the *Lightning Platform Canvas Developer's Guide*.

Note: For a canvas app to appear in a console, you must add it to the console as a custom console component.

When developing a canvas app, and you want to include functionality from the Salesforce Console Integration Toolkit, do the following:

1. Include the console integration toolkit API in `index.jsp`.
2. If your console has an allowlist for domains, add the domain of your canvas app to the allowlist.
3. Call `Sfdc.canvas.client.signedrequest()` to store the signed request needed by the console integration toolkit API. For example, if the Lightning Platform Canvas method of authentication is a signed request, do the following:

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

   If the Lightning Platform Canvas method of authentication is OAuth, do the following in the callback function used to get the context as shown in "Getting Context in Your Canvas App" in the *Lightning Platform Canvas Developer's Guide*:

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

Consider the following when working with the Salesforce Console Integration Toolkit and canvas apps:

- The console integration toolkit 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 *Lightning Platform Canvas Developer's Guide*.

SEE ALSO:

Salesforce Canvas Developer Guide: Getting Context in Your Canvas App

Salesforce Help: Add Console Components to Apps in Salesforce Classic

Salesforce Help: Allow Domains for a Salesforce Console in Salesforce

Salesforce Console Integration Toolkit Best Practices

Salesforce recommends that you adhere to a few best practices as you use the Salesforce Console Integration Toolkit.

- Many of the methods in the Salesforce Console Integration Toolkit are asynchronous and return their results using a callback method. We recommend that you refer to the documentation for each method to understand the information for each response.
- Errors generated by the Salesforce Console Integration Toolkit are typically emitted in a way that doesn't halt JavaScript processing. Therefore, we recommend that you use a tool such as Firebug for Firefox to monitor the JavaScript console and to help you debug your code.
- Due to third-party cookie restrictions in modern web browsers, Visualforce pages can't load in Salesforce Classic console apps when third-party cookies are disabled. See Visualforce Limitations in Salesforce Classic When Third-Party Cookies are Blocked.
- To display Visualforce pages properly in the Salesforce Console, we recommend you:
  - Accept the default setting `showHeader="true"` and set `sidebar="false"` on the `apex:page` tag.
  - Set `Behavior` on custom buttons and links that include methods from the toolkit to display in an existing window without a sidebar or header. For more information, see Define Custom Buttons and Links" in the Salesforce online help.
- When using Firefox, we recommend that you don't call `closeTab()` on a tab with an active alert box because the browser may not load properly.
- Duplicate tabs might open when users initiate methods with invalid URLs. We recommend that you check URLs for validity before you include them in methods.
- To prevent `External Page` from displaying as a tab name, we recommend that you specify the `tabLabel` argument on methods such as `openPrimaryTab()` and `openSubtab()`.
- To enable the toolkit for third-party domains, add the domains to the allowlist of the Salesforce console.
- The Salesforce Console Integration Toolkit methods don't work in nested iframes. For example, if you use a custom quick action in a feed, the methods still work as expected because the feed is in a single iframe. But if Development Mode is also enabled in your org, the methods no longer work because the iframe of the feed is nested inside the Development Mode iframe.

Methods for Salesforce Classic

If your org is using Salesforce Classic, use Salesforce Console Integration Toolkit methods.

IN THIS SECTION:

Methods for Primary Tabs and Subtabs

A Salesforce console displays Salesforce pages as primary tabs or subtabs. A primary tab displays the main item to work on, such as an account. A subtab displays related items, such as an account's contacts or opportunities.

IN THIS SECTION:

focusSubtabByNameAndPrimaryTabName()

Focuses the browser on a subtab that is already open with the specified name and primary tab name. This method is only available in API version 22.0 or later.

generateConsoleUrl()

Generates a URL to a tab, or group of related tabs, in the Salesforce console. If any tabs include external URLs, then add the external URLs to the console's allowlist so that they can display correctly.This method is only available in API version 28.0 or later.

getEnclosingPrimaryTabId()

Returns the ID of the current primary tab. This method works within a primary tab or subtab, not within the navigation tab or custom console components. This method is only available in API version 20.0 or later.

getEnclosingPrimaryTabObjectId()

Returns the object ID of the current primary tab, which contains a subtab. For example, a case ID or account ID. This method works within a primary tab or subtab. This method is only available in API version 24.0 or later.

getEnclosingTabId()

Returns the ID of the tab that contains the current Visualforce page, which may be a primary tab or subtab. This method will work if the call is made within a component enclosed within a subtab. This method is only available in API version 20.0 or later.

getFocusedPrimaryTabId()

Returns the ID of the primary tab on which the browser is focused. This method is only available in API version 25.0 or later.

getFocusedPrimaryTabObjectId()

Returns the object ID of the primary tab on which the browser is focused. This method is only available in API version 25.0 or later.

getFocusedSubtabId()

Returns the ID of the subtab on which the browser is focused. For example, a case ID or account ID. This method is only available in API version 25.0 or later.

getFocusedSubtabObjectId()

Returns the object ID of the subtab on which the browser is focused. For example, a case ID or account ID. This method is only available in API version 24.0 or later.

getPageInfo()

Returns page information for the specified tab after its content has loaded. If the tab ID is null, it returns page information for the enclosing primary tab or subtab. Note that to get the page information from a custom console component, a `tabId` must be passed as the first parameter to this method.This method is only available in API version 26.0 or later.

getPrimaryTabIds()

Returns all of the IDs of open primary tabs. This method is only available in API version 26.0 or later.

getSubtabIds()

Returns all of the IDs of the subtabs on the primary tab specified by a primary tab ID. If the primary tab ID is null, it returns the IDs of the subtabs on the current primary tab. This method can only be called from a custom console component or a detail page overwritten by a Visualforce page. This method is only available in API version 26.0 or later.

getTabLink()

Retrieves the URL to a tab, or group of related tabs, from the Salesforce console. This method is only available in API version 28.0 or later.

isInConsole()

Determines if the page is in the Salesforce console. This method is only available in API version 22.0 or later.

onEnclosingTabRefresh()

Registers a function to call when the enclosing tab refreshes. This method is only available in API version 24.0 or later.

onFocusedPrimaryTab()

Registers a function to call when the focus of the browser changes to a different primary tab. This method is only available in API version 25.0 or later.

onFocusedSubtab()

Registers a function to call when the focus of the browser changes to a different subtab. This method is only available in API version 24.0 or later.

onTabSave()

Registers and calls a callback method when a user clicks **Save** in a subtab's Unsaved Changes dialog box. When using this method, call `setTabUnsavedChanges()` in the callback method. This notifies the console that the custom save operation completed. In the call to `setTabUnsavedChanges()`, pass the first parameter as `false` to indicate a successful save or `true` to indicate an unsuccessful save. This method is only available in API version 28.0 or later.

openConsoleUrl()

Opens a URL created by the `generateConsoleUrl()` method (a URL to a tab, or group of related tabs, in the Salesforce console). This method is only available in API version 28.0 or later.

openPrimaryTab()

Opens a new primary tab to display the content of the specified URL, which can be relative or absolute. You can also override an existing tab. This method is only available in API version 20.0 or later.

openSubtab()

Opens a new subtab (within a primary tab) that displays the content of a specified URL, which can be relative or absolute. You can also override an existing subtab. Use to open a new subtab on a primary tab via the primary tab's ID. This method is only available in API version 20.0 or later.

openSubtabByPrimaryTabName()

Opens a new subtab (within a primary tab) that displays the content of a specified URL, which can be relative or absolute. You can also override an existing subtab. Use to open a new subtab on a primary tab via the primary tab's name. This method is only available in API version 22.0 or later.

refreshPrimaryTabById()

Refreshes a primary tab specified by ID, including its subtabs. This method can't refresh subtabs with URLs to external pages or Visualforce pages. This method is only available in API version 22.0 or later.

refreshPrimaryTabByName()

Refreshes a primary tab specified by name, including its subtabs. This method can't refresh subtabs with URLs to external pages or Visualforce pages. This method is only available in API version 22.0 or later.

refreshSubtabById()

Refreshes a subtab with the last known URL with a specified ID. This method can't refresh a subtab if the last known URL is an external page or a Visualforce page. This method is only available in API version 22.0 or later.

refreshSubtabByNameAndPrimaryTabId()

Refreshes a subtab with the last known URL with the specified name and primary tab ID. This method can't refresh a subtab if the last known URL is an external page or a Visualforce page. This method is only available in API version 22.0 or later.

refreshSubtabByNameAndPrimaryTabName()

Refreshes a subtab with the last known URL with the specified name and primary tab name. This method can't refresh a subtab if the last known URL is an external page or a Visualforce page. This method is only available in API version 22.0 or later.

reopenLastClosedTab()

Reopens the last closed primary tab, and any of its subtabs that were open, the moment it was closed. This method is only available in API version 35.0 or later.

resetSessionTimeOut()

Resets a session timeout for a console app. This method ensures that users can continue working on Visualforce pages without being prompted to log back in to the console when they return to a console tab or console component. This method is only available in API version 24.0 or later.

setTabUnsavedChanges()

Sets the unsaved changes icon (*) on subtabs to indicate unsaved data. This method is only available in API version 23.0 or later.

setTabIcon()

Sets an icon on the specified tab. If a tab is not specified, the icon is set on the enclosing tab. Use this method to customize a tab's icon. This method is only available in API version 28.0 or later.

setTabLink()

Sets a console tab's URL attribute to the location of the tab's content. Use this method to generate secure console URLs when users navigate to tabs displaying content outside of the Salesforce domain.This method is only available in API version 28.0 or later.

setTabStyle()

Sets a cascading style sheet (CSS) on the specified tab. If a tab is not specified, the CSS is set on the enclosing tab. Use this method to customize a tab's look and feel. This method is only available in API version 28.0 or later.

setTabTextStyle()

Sets a cascading style sheet (CSS) on a specified tab's text. If a tab is not specified, the CSS is set on the enclosing tab's text. Use this method to customize a tab's text style. This method is only available in API version 28.0 or later.

setTabTitle()

Sets the title of a primary tab or subtab. This method is only available in API version 20.0 or later.

`closeTab()`

Closes a specified primary tab or subtab. Keep in mind that closing the first tab in a primary tab closes the primary tab itself. This method is only available in API version 20.0 or later.

Note: The user interface and API behave different for pinned primary tabs. In the UI, when a primary tab is pinned, you can close subtabs using your mouse. However, in the API, if the primary tab is pinned, you can't close its subtabs.

Syntax

```
sforce.console.closeTab(id:String, (optional) callback:Function)
```

Arguments

Name	Type	Description
`id`	string	ID of the primary tab or subtab to close.
`callback`	function	For API version 35.0 or later, the JavaScript method that's called upon completion of the method.

Sample Code API 20.0 or Later–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testCloseTab();return false">
```

```
            Click here to close this tab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testCloseTab() {
            //First find the ID of the current tab to close it
            sforce.console.getEnclosingTabId(closeSubtab);
        }

        var closeSubtab = function closeSubtab(result) {
            //Now that we have the tab ID, we can close it
            var tabId = result.id;
            sforce.console.closeTab(tabId);
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

None

Sample Code API Version 35.0 or Later–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testCloseTab();return false">
            Click here to close this tab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
         var callback = function () {
             if (result.error) {
          alert("Error message is " + result.error);
             }
         };
        function testCloseTab() {
            //First find the ID of the current tab to close it
            sforce.console.getEnclosingTabId(closeSubtab);
         }

        var closeSubtab = function closeSubtab(result) {
            //Now that we have the tab ID, we can close it
            var tabId = result.id;
            sforce.console.closeTab(tabId, callback);
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	`true` if the tab was re-opened, `false` otherwise.
`error`	string	Error message if the tab couldn't be closed.

Tip: When using Firefox, we recommend that you don't call `closeTab()` on a tab with an active alert box because the browser may not load properly.

`disableTabClose()`

Prevents a user from closing a tab or a subtab. If the ID parameter doesn't specify a tab, the enclosing tab is used. You can also use this method to re-enable a tab that has been disabled. Available in API version 36.0 or later.

Note:

- If you disable subtabs from closing, the primary tab is also disabled from closing.
- If a record is deleted whose primary tab is disabled, the tab is forcibly closed.
- If a record is deleted whose subtab is disabled, the subtab is not closed.

Syntax

```
sforce.console.disableTabClose(disable:boolean, (optional) tabId:String, (optional)
callback:Function)
```

Arguments

Name	Type	Description
`disable`	boolean	Specifies whether to disable the tab. If `true`, the user can't close the tab. If `false`, the user can close the tab.
`tabId`	string	The tabId for the tab to enable or disable. Use `false` to automatically select the enclosing tab or subtab without needing to specify a tabId. The enclosing tabId can't be inferred when this call is made from outside a sidebar component. For example, if you call this method from a footer widget, specify the tabId. If the tabId is for a "Details" subtab of a primary tab, the primary tabId is used instead.
`callback`	function	JavaScript method that's called upon completion of the method. The callback is passed a response object. See response information below.

Sample Code–Visualforce

```
<apex:page >
<html>
  <head>
    <title>Disable close Tab on Load</title>

    <!-- Service Console integration API library -->
    <script src="/support/console/63.0/integration.js"></script>

    <!-- Callback functions to handle tab events -->
    <script type="text/javascript">

      function displayResultsCallback(result){
        var resDiv = document.getElementById("eventResults");
        resDiv.innerHTML = JSON.stringify(result);
      }

      // For use within a tab's sidebar (you don't need tab ID)

      function testDisableTabCloseTrueWithoutId() {
        sforce.console.disableTabClose(true, false, displayResultsCallback);
      }

      function testDisableTabCloseFalseWithoutId() {
        sforce.console.disableTabClose(false, false, displayResultsCallback);
      }

      // For use anywhere (you need the tab ID)

      // Note: Your tab ID might be different than the one used here.
      //       You can get the tab ID many different ways,
      //       including sforce.console.getEnclosingTabId().
      //       See the documentation for details.
      function testDisableTabCloseTrueWithId() {
        var tabId = window.prompt("Enter the tab ID","scc-pt-0");
        sforce.console.disableTabClose(true, tabId, displayResultsCallback);
      }

      function testDisableTabCloseFalseWithId() {
        var tabId = window.prompt("Enter the tab ID","scc-pt-0");
        sforce.console.disableTabClose(false, tabId, displayResultsCallback);
      }

    </script>
  </head>

  <body>
    <h1>Disable Tab Close Examples</h1>
    <br/><br/>

    <h2>API Callback Result</h2>
    <br/>

    <code><div id="eventResults" /></code>
```

```
    <br/>

    <h2>With No Tab ID</h2>
    <p>The tab ID will be auto-detected by context, or the event will fail.</p>

    <ul>
    <li><a href="#" onClick="testDisableTabCloseTrueWithoutId();return false;">
    Disable closing for the enclosing tab</a></li>

    <li><a href="#" onClick="testDisableTabCloseFalseWithoutId();return false;">
    Re-enable closing for the enclosing tab</a></li>
    </ul>

    <h2>With Tab ID Provided</h2>
    <p>When the event context doesn't have a detectable tab ID, you can
    supply it yourself.</p>

    <ul>
    <li><a href="#" onClick="testDisableTabCloseTrueWithId();return false;">
    Disable closing for a specific tab (via tab ID)</a></li>

    <li><a href="#" onClick="testDisableTabCloseFalseWithId();return false;">
    Re-enable closing for a specific tab (via tab ID)</a></li>
    </ul>

  </body>
</html>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	`true` if the action completed successfully, `false` otherwise.
`message`	string	If the action completed successfully, message contains the affected tabId. If the action failed, message contains the error message.

`focusPrimaryTabById()`

Focuses the browser on a primary tab that is already open with the specified ID. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.focusPrimaryTabById(id:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`id`	string	ID of the primary tab to go to.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testFocusPrimaryTabById();return false">
         Click here to go to an open primary tab by id</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testFocusPrimaryTabById() {
            //Get the value for 'scc-pt-0' from the openPrimaryTab method
            //This value is for example purposes only
            var primaryTabId = 'scc-pt-0';
            sforce.console.focusPrimaryTabById(primaryTabId, focusSuccess);
        }

        var focusSuccess = function focusSuccess(result) {
            //Report whether going to the open primary tab was successful
            if (result.success == true) {
                alert('Going to the primary tab was successful');
            } else {
                alert('Going to the primary tab was not successful');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if going to the primary tab was successful; `false` if going to the primary tab wasn't successful.

focusPrimaryTabByName()

Focuses the browser on a primary tab that is already open with the specified name. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.focusPrimaryTabByName(name:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
name	string	Name of the primary tab to go to.
callback	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testFocusPrimaryTabByName();return false">
         Click here to go to a primary tab by name</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testFocusPrimaryTabByName() {
            //Get the value for 'myPrimaryTab' from the openPrimaryTab method
            //This value is for example purposes only
            var primaryTabName = 'myPrimaryTab';
            sforce.console.focusPrimaryTabByName(primaryTabName, focusSuccess);
        }

        var focusSuccess = function focusSuccess(result) {
            //Report whether going to the primary tab was successful
            if (result.success == true) {
                alert('Going to the primary tab was successful');
            } else {
                alert('Going to the Primary tab was not successful');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if going to the primary tab was successful; `false` if going to the primary tab wasn't successful.

`focusSidebarComponent()`

Focuses the browser on a sidebar component. Use this method to focus on a component with the tab or accordion sidebar style.

Syntax

```
sforce.console.focusSidebarComponent(componentInfo:string (optional)tabId:string,
callback:Function)
```

Arguments

Name	Type	Description
`componentInfo`	string	The JSON object that represents the component to focus on. This argument must include one of the following forms: Unambiguous types: • `{componentType: 'CASE_EXPERT_WIDGET' }` • `{componentType: 'FILES_WIDGET' }` • `{componentType: 'HIGHLIGHTS_PANEL' }` • `{componentType: 'KNOWLEDGE_ONE'}` • `{componentType: 'MILESTONE_WIDGET' }` • `{componentType: 'TOPICS_WIDGET' }` • `{componentType: 'VISUALFORCE' }` Types that require additional parameters: • `{componentType: 'CANVAS', canvasAppId: '09Hxx0000000001'}` • `{componentType: 'RELATED_LIST', listName: 'Solution'}` • `{componentType: 'LOOKUP', fieldName: 'Account'}` • `{componentType: 'VISUALFORCE', pageName: 'VF1'}`
`tabId`	string	The ID of the tab on which to focus the browser.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var callback = function (result) {}
            if(result.success){
            alert('Congratulations!');
        }else{
            alert('Something is wrong!');
        }
    };
        function focusKnowledgeComponent() {
            sforce.console.focusSidebarComponent(JSON.stringify({componentType:
            'KNOWLEDGE_ONE'}),"scc-st-2", callback);
        }
    </script>
   <A HREF="#" onClick="focusSidebarComponent(); return false">Focus Knowledge Component</A>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if focusing the sidebar component was successful; `false` otherwise.

`focusSubtabById()`

Focuses the browser on a subtab that is already open with the specified ID. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.focusSubtabById(id:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`id`	string	ID of the subtab to go to.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">
```

```
     <A HREF="#" onClick="testFocusSubtabById();return false">
         Click here to go to a subtab by id</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testFocusSubtabById() {
            //Get the value for 'scc-st-0' from the openSubtab method
            //This value is for example purposes only
            var subtabId = 'scc-st-0';
            sforce.console.focusSubtabById(subtabId, focusSuccess);
        }

        var focusSuccess = function focusSuccess(result) {
            //Report whether going to the subtab was successful
            if (result.success == true) {
                alert('Going to the subtab was successful');
            } else {
                alert('Going to the subtab was not successful');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if going to the subtab was successful; `false` if going to the subtab wasn't successful.

`focusSubtabByNameAndPrimaryTabId()`

Focuses the browser on a subtab that is already open with the specified name and primary tab ID. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.focusSubtabByNameAndPrimaryTabId(name:String,
primaryTabId:String,(optional)callback:Function)
```

Arguments

Name	Type	Description
`name`	string	Name of the subtab to go to.
`primaryTabId`	string	ID of the primary tab in which the subtab opened.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testFocusSubtabByNameAndPrimaryTabId();return false">
         Click here to go to a subtab by name and primary tab ID</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testFocusSubtabByNameAndPrimaryTabId() {
            //Get the values for 'mySubtab' and 'scc-pt-0' from the openSubtab method
            //These values are for example purposes only
            var subtabName = 'mySubtab';
            var primaryTabId = 'scc-pt-0';
            sforce.console.focusSubtabByNameAndPrimaryTabId(subtabName, primaryTabId,
focusSuccess);
        }

        var focusSuccess = function focusSuccess(result) {
            //Report whether going to the subtab was successful
            if (result.success == true) {
                alert('Going to the subtab was successful');
            } else {
                alert('Going to the subtab was not successful');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if going to the subtab was successful; `false` if going to the subtab wasn't successful.

focusSubtabByNameAndPrimaryTabName()

Focuses the browser on a subtab that is already open with the specified name and primary tab name. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.focusSubtabByNameAndPrimaryTabName(name:String,
primaryTabName:String,(optional)callback:Function)
```

Arguments

Name	Type	Description
name	string	Name of the subtab to go to.
primaryTabName	string	Name of the primary tab in which the subtab opened.
callback	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testFocusSubtabByNameAndPrimaryTabName();return false">
         Click here to go to a subtab by name and primary tab name</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testFocusSubtabByNameAndPrimaryTabName() {
            //Get the value for 'mySubtab' and 'myPrimaryTab' from the openSubtab method
            //These values are for example purposes only
            var subtabName = 'mySubtab';
            var primaryTabName = 'myPrimaryTab';
            sforce.console.focusSubtabByNameAndPrimaryTabName(subtabName, primaryTabName,
 focusSuccess);
        }

        var focusSuccess = function focusSuccess(result) {
            //Report whether going to the subtab was successful
            if (result.success == true) {
                alert('Going to the subtab was successful');
            } else {
                alert('Going to the subtab was not successful');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if going to the subtab was successful; `false` if going to the subtab wasn't successful.

`generateConsoleUrl()`

Generates a URL to a tab, or group of related tabs, in the Salesforce console. If any tabs include external URLs, then add the external URLs to the console's allowlist so that they can display correctly.This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.generateConsoleUrl(urls:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`urls`	string	An array of URLs. The first URL is a primary tab and subsequent URLs are subtabs. Note that the last URL is the subtab on which the console is focused. These URLs can be standard Salesforce URLs or relative URLs.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <A HREF="#" onClick="testGenerateConsoleURL();return false">
        Click here to generate a console URL</A>

    <script type="text/javascript">
        function showConsoleUrl(result) {
            alert(result.consoleUrl);
         }
        function testGenerateConsoleURL() {
            sforce.console.generateConsoleUrl([/apex/pagename, /entityId,
www.externalUrl.com, Standard Salesforce Url/entityId], showConsoleUrl);        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`consoleUrl`	string	Console URL that represents the array of URLs passed into Salesforce.
`success`	boolean	`true` if the URL was generated successfully, `false` if otherwise.
`callback`	function	JavaScript method that's called upon completion of the method.

`getEnclosingPrimaryTabId()`

Returns the ID of the current primary tab. This method works within a primary tab or subtab, not within the navigation tab or custom console components. This method is only available in API version 20.0 or later.

Syntax

```
sforce.console.getEnclosingPrimaryTabId((optional)callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testCloseTab();return false">
         Click here to close this primary tab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testCloseTab() {
            //First find the ID of the current primary tab to close it
            sforce.console.getEnclosingPrimaryTabId(closeSubtab);
        }

        var closeSubtab = function closeSubtab(result) {
            //Now that we have the primary tab ID, we can close it
            var tabId = result.id;
            sforce.console.closeTab(tabId);
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`id`	string	The ID of the current primary tab that contains this tab.

`getEnclosingPrimaryTabObjectId()`

Returns the object ID of the current primary tab, which contains a subtab. For example, a case ID or account ID. This method works within a primary tab or subtab. This method is only available in API version 24.0 or later.

Syntax

```
sforce.console.getEnclosingPrimaryTabObjectId((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testGetEnclosingPrimaryTabObjectId();">
           Click here to get enclosing primary tab object ID</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetEnclosingPrimaryTabObjectId() {
            sforce.console.getEnclosingPrimaryTabObjectId(showObjectId);
        }
            var showObjectId = function showObjectId(result) {
                // Display the object ID
            alert ('Object ID: ' + result.id);
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`id`	string	The ID of the current primary tab that contains this subtab.
`success`	boolean	`true` if returning the enclosing primary tab was successful; `false` if returning the enclosing primary tab wasn't successful.

`getEnclosingTabId()`

Returns the ID of the tab that contains the current Visualforce page, which may be a primary tab or subtab. This method will work if the call is made within a component enclosed within a subtab. This method is only available in API version 20.0 or later.

Syntax

```
sforce.console.getEnclosingTabId()
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testCloseTab();return false">
        Click here to close this tab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testCloseTab() {
            //First find the ID of the current tab to close it
            sforce.console.getEnclosingTabId(closeSubtab);
        }

        var closeSubtab = function closeSubtab(result) {
            //Now that we have the tab ID, we can close it
            var tabId = result.id;
            sforce.console.closeTab(tabId);
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
id	string	The ID of the current primary tab or subtab.

getFocusedPrimaryTabId()

Returns the ID of the primary tab on which the browser is focused. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.getFocusedPrimaryTabId((optional) callback:Function)
```

Arguments

Name	Type	Description
callback	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

     <A HREF="#" onClick="testGetFocusedPrimaryTabId();return false">
         Click here to get the focused primary tab ID</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetFocusedPrimaryTabId() {
            sforce.console.getFocusedPrimaryTabId(showTabId);
        }
        var showTabId = function showTabId(result) {
            //Display the tab ID
            alert('Tab ID: ' + result.id);
        };

  </script>

</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
id	string	The ID of the primary tab on which the browser is focused. If no primary tab is open, the ID is null.

Name	Type	Description
`success`	boolean	`true` if returning the primary tab ID on which the browser is focused was successful; `false` if returning the primary tab ID on which the browser is focused wasn't successful.

`getFocusedPrimaryTabObjectId()`

Returns the object ID of the primary tab on which the browser is focused. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.getFocusedPrimaryTabObjectId((optional) callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

     <A HREF="#" onClick="testGetFocusedPrimaryTabObjectId();return false">
         Click here to get the focused primary tab object ID</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetFocusedPrimaryTabObjectId() {
            sforce.console.getFocusedPrimaryTabObjectId(showObjectId);
        }
        var showObjectId = function showObjectId(result) {
            //Display the object ID
            alert('Object ID: ' + result.id);
        };

  </script>

</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`id`	string	The object ID of the primary tab on which the browser is focused. If there is no primary tab open, the ID is null.
`success`	boolean	`true` if returning the primary tab object ID on which the browser is focused was successful; `false` if returning the primary tab object ID on which the browser is focused wasn't successful.

`getFocusedSubtabId()`

Returns the ID of the subtab on which the browser is focused. For example, a case ID or account ID. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.getFocusedSubtabId((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testGetFocusedSubtabId();">
            Click here to get the ID of the focused subtab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetFocusedSubtabId() {
            sforce.console.getFocusedSubtabId(showTabId);
        }
        var showTabId = function showTabId(result) {
                // Display the tab ID
            alert ('Tab ID: ' + result.id);
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`id`	string	The ID of the subtab on which the browser is focused. If no subtab is open, the ID is null.
`success`	boolean	`true` if returning the ID of the focused subtab was successful; `false` if returning the ID of the focused subtab wasn't successful.

`getFocusedSubtabObjectId()`

Returns the object ID of the subtab on which the browser is focused. For example, a case ID or account ID. This method is only available in API version 24.0 or later.

Syntax

```
sforce.console.getFocusedSubtabObjectId((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testGetFocusedSubtabObjectId();">
           Click here to get the object ID of the focused subtab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetFocusedSubtabObjectId() {
            sforce.console.getFocusedSubtabObjectId(showObjectId);
        }
            var showObjectId = function showObjectId(result) {
                // Display the object ID
            alert ('Object ID: ' + result.id);
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`id`	string	The object ID of the subtab on which the browser is focused. If no subtab is open, the ID is null.
`success`	boolean	`true` if returning the object ID of the focused subtab was successful; `false` if returning the object ID of the focused subtab wasn't successful.

`getPageInfo()`

Returns page information for the specified tab after its content has loaded. If the tab ID is null, it returns page information for the enclosing primary tab or subtab. Note that to get the page information from a custom console component, a `tabId` must be passed as the first parameter to this method.This method is only available in API version 26.0 or later.

Syntax

```
sforce.console.getPageInfo(tabId:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`tabId`	string	ID of the tab from which page information is returned.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testGetPageInfo();return false">
        Click here to get page information</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetPageInfo() {
            //Get the page information of 'scc-pt-1'
           //This value is for example purposes only
             var tabId = 'scc-pt-1';
              sforce.console.getPageInfo(tabId , showPageInfo);
        }

        var showPageInfo = function showPageInfo(result) {
            alert('Page Info: ' + result.pageInfo);
            };
  </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description

`pageInfo`	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://MyDomainNamemy.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://MyDomainNamemy.salesforce.com/001x0000003DGQR", "objectId":"001x0000003DGQR","objectName":"Acme Person Account", "object":"Account", "contactId":"003D000000QOMqg", "personAccount":true}`

`callback`	function	JavaScript method that's called upon completion of the method.

`getPrimaryTabIds()`

Returns all of the IDs of open primary tabs. This method is only available in API version 26.0 or later.

Syntax

```
sforce.console.getPrimaryTabIds((optional) callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testGetPrimaryTabIds();return false">
        Click here to get the primary tab IDs</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetPrimaryTabIds() {
```

```
            sforce.console.getPrimaryTabIds(showTabId);
        }

        var showTabId = function showTabId(result) {
            //Display the primary tab IDs
            alert('Primary Tab IDs: ' + result.ids);
            };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`ids`	string	An array of open primary tab IDs , in order of appearance.
`success`	boolean	`true` if returning the IDs of open primary tabs was successful; `false` if returning the IDs of open primary tabs wasn't successful.

`getSubtabIds()`

Returns all of the IDs of the subtabs on the primary tab specified by a primary tab ID. If the primary tab ID is null, it returns the IDs of the subtabs on the current primary tab. This method can only be called from a custom console component or a detail page overwritten by a Visualforce page. This method is only available in API version 26.0 or later.

Syntax

```
sforce.console.getSubtabIds( (optional) primaryTabId:String, (optional) callback:Function)
```

Arguments

Name	Type	Description
`primaryTabId`	string	ID of the primary tab from which the subtab IDs are returned.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testGetSubtabIds();return false">
        Click here to get the subtab IDs</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testGetSubtabIds() {
```

```
            //Get the subtabs of the primary tab 'scc-pt-0'
           //This value is for example purposes only
             var primaryTabId = 'scc-pt-0';
              sforce.console.getSubtabIds(primaryTabId , showTabId);
        }

        var showTabId = function showTabId(result) {
            //Display the subtab IDs
            alert('Subtab IDs: ' + result.ids);
            };
  </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`ids`	string	An array of open subtab IDs.
`success`	boolean	`true` if firing the event was successful; `false` if firing the event wasn't successful.

`getTabLink()`

Retrieves the URL to a tab, or group of related tabs, from the Salesforce console. This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.getTabLink(level:String, (optional)tabId:String,
(optional)callback:Function)
```

Arguments

Name	Type	Description
`level`	string	Level that matches one of the Link to Share options in the Salesforce console user interface. The options are: • All primary tabs and subtabs — `sforce.console.TabLink.PARENT_AND_CHILDREN`. • Only the specified tab — `sforce.console.TabLink.TAB_ONLY` • A standard Salesforce URL — `sforce.console.TabLink.SALESFORCE_URL`
`tabId`	string	Optional tab ID of the tab from which you're retrieving the URL. If you do not pass a tab ID, the URL to the current tab is returned.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <A HREF="#" onClick="getEnclosingPrimaryTabId();return false">
        Click here to get tab link</A>

    <script type="text/javascript">
        var getEnclosingPrimaryTabId = function getEnclosingPrimaryTabId() {
            sforce.console.getEnclosingPrimaryTabId(getTabLink);
        }
        var getTabLink = function getTabLink(result) {
           sforce.console.getTabLink(sforce.console.TabLink.PARENT_AND_CHILDREN, result.id,
 showTabLink);
        }
        var showTabLink = function showTabLink(result) {
            var link = result.tabLink;
            };
  </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`tabLink`	string	The retrieved URL.
`success`	boolean	`true` if the link was retrieved successfully, `false` if retrieving was unsuccessful.
`callback`	function	JavaScript method that's called upon completion of the method.

`isInConsole()`

Determines if the page is in the Salesforce console. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.isInConsole()
```

Arguments

None

Sample Code–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testIsInConsole();return false">
         Click here to check if the page is in the Service Cloud console</A>
```

```
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testIsInConsole() {
            if (sforce.console.isInConsole()) {
                  alert('in console');
               } else {
                  alert('not in console');
            }
        }
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

Returns `true` if the page is in the Salesforce console; `false` if the page is not in the Salesforce console.

onEnclosingTabRefresh()

Registers a function to call when the enclosing tab refreshes. This method is only available in API version 24.0 or later.

Syntax

```
sforce.console.onEnclosingTabRefresh(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when the enclosing tab refreshes.

Sample Code–Visualforce

```
<apex:page>

    <apex:includeScript value="/support/console/63.0/integration.js"/>

    <script type="text/javascript">
        var eventHandler = function eventHandler(result) {
            alert('Enclosing tab has refreshed:' + result.id
          + 'and the object Id is:' + result.objectId);
        };
            sforce.console.onEnclosingTabRefresh(eventHandler);
    </script>
</apex:page>
```

Event Handler Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`id`	string	The ID of the refreshed tab.
`objectId`	string	The object ID of the refreshed tab or null if no object exists.

`onFocusedPrimaryTab()`

Registers a function to call when the focus of the browser changes to a different primary tab. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.onFocusedPrimaryTab(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when the focus of the browser changes to a different primary tab.

Sample Code–Visualforce

```
<apex:page>

    <apex:includeScript value="/support/console/63.0/integration.js"/>

    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('Focus changed to a different primary tab. The primary tab ID is:'
        + result.id + 'and the object Id is:' + result.objectId);
        };
        sforce.console.onFocusedPrimaryTab(eventHandler);
    </script>
</apex:page>
```

Event Handler Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`id`	string	The ID of the primary tab on which the browser is focused.

Name	Type	Description
objectId	string	The object ID of the primary tab on which the browser is focused or null if no object exists.

onFocusedSubtab()

Registers a function to call when the focus of the browser changes to a different subtab. This method is only available in API version 24.0 or later.

Syntax

```
sforce.console.onFocusedSubtab(eventHandler:Function)
```

Arguments

Name	Type	Description
eventHandler	function	JavaScript method called when the focus of the browser changes to a different subtab.

Sample Code–Visualforce

```
<apex:page>

    <apex:includeScript value="/support/console/63.0/integration.js"/>

    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('Focus changed to a different subtab. The subtab Id is:'
        + result.id + 'and the object Id is:' + result.objectId);
        };
        sforce.console.onFocusedSubtab(eventHandler);
    </script>
</apex:page>
```

Event Handler Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
id	string	The ID of the subtab on which the browser is focused.
objectId	string	The object ID of the subtab on which the browser is focused or null if no object exists.

onTabSave()

Registers and calls a callback method when a user clicks **Save** in a subtab's Unsaved Changes dialog box. When using this method, call `setTabUnsavedChanges()` in the callback method. This notifies the console that the custom save operation completed. In the call to `setTabUnsavedChanges()`, pass the first parameter as `false` to indicate a successful save or `true` to indicate an unsuccessful save. This method is only available in API version 28.0 or later.

Registering a callback method affects the user interface. When no save handler is registered, the user is presented with two options when closing a subtab with unsaved changes: **Continue** or **Cancel**. When a save handler is registered, the user is presented with three options when closing the subtab: **Save**, **Don't Save**, or **Cancel**. In this scenario, the callback method registered is called when the user chooses **Save**.

Important: When using `onTabSave()` with `setTabUnsavedChanges()`:

- Calling `sforce.console.setTabUnsavedChanges(false,...)` closes the specified subtab. We recommend placing the call to `sforce.console.setTabUnsavedChanges()` at the end of the callback method, as any subsequent save logic might not execute.
- `onTabSave()` works only on subtabs or their sidebar components. It doesn't work on primary tabs.
- *Not* calling `sforce.console.setTabUnsavedChanges()` will have a severe effect on the user interface. For example, closing a primary tab with a subtab for which `sforce.console.setTabUnsavedChanges()` has not been called prevents a `Saving...` modal dialog box from closing.
- Any callback passed to `sforce.console.setTabUnsavedChanges()` will not execute if the specified tab saves successfully and closes.

Note: Calling `onTabSave()` from a custom console component prevents that component from refreshing when saving the subtab. " in the Salesforce online help.

Syntax

```
sforce.console.onTabSave(callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called to handle the save operation.

Sample Code–Visualforce

```
<apex:page>
    <a href="#" onClick="testOnTabSave();return false">
         Click here to register save handler</a>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testOnTabSave() {
            sforce.console.onTabSave(handleSave);
        }
    var handleSave = function handleSave(result) {
        alert('save handler called from tab with id ' + result.id +
```

```
            ' and objectId ' + result.objectId);
        //Perform save logic here

        //Mark tab as 'clean'
        sforce.console.setTabUnsavedChanges(false, undefined, result.id);
    };
  </script>
</apex:page>
```

Response

Name	Type	Description
`id`	string	ID of the subtab being saved.
`objectId`	string	Object ID of the subtab being saved, if applicable; null otherwise.

`openConsoleUrl()`

Opens a URL created by the `generateConsoleUrl()` method (a URL to a tab, or group of related tabs, in the Salesforce console). This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.openConsoleUrl(id:String, consoleUrl:URL, active:Boolean,
(optional)tabLabels:String, (optional)tabNames:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`id`	string	ID of the console tab to override. If the ID corresponds to an existing primary tab, then the existing primary tab is redirected to the given URL because the console prevents duplicate tabs. Use null to create a new primary tab.
`consoleUrl`	string	Console URL that represents the array of URLs passed into Salesforce.
`active`	boolean	If `true`, the opened primary tab displays immediately. If `false`, the opened primary tab displays in the background and the current tab maintains focus.
`tabLabels`	string	Optional array of labels of the opened primary tab or subtabs. The order in which the tabs appear in the console URL should match the order of the labels that appear in the array. If you do not want to set the labels of tabs, use an empty string (`''`).
`tabNames`	string	Optional array of names of the opened primary and subtabs. The order in which the tabs appear in the console URL should match the order of the names that appear in the array. If you do not want to set the names of tabs, use an empty string (`''`).
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <A HREF="#" onClick="testGenerateConsoleURL();return false">
        Click here to open a console URL</A>

    <script type="text/javascript">
        var generateConsoleUrl = function testGenerateConsoleURL() {
            sforce.console.generateConsoleUrl([/apex/pagename, /entityId,
www.externalUrl.com, Standard Salesforce Url/entityId], showConsoleUrl);
         }
        var openConsoleUrl = function showConsoleUrl(result) {
            sforce.console.openConsoleUrl(null, result.consoleUrl, true, ['Apex', '',
'Salesforce', ''], ['', '', 'externalUrl', ''])
         }
  </script>
</apex:page>
```

Note: This example shows that if you want to set a label or name, you must set the other values to empty string (`''`).

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the console URL was opened successfully, `false` otherwise.

`openPrimaryTab()`

Opens a new primary tab to display the content of the specified URL, which can be relative or absolute. You can also override an existing tab. This method is only available in API version 20.0 or later.

- If the ID corresponds to an existing primary tab, the existing tab is redirected to the given URL because the Salesforce console prevents duplicate tabs.
- If the URL is to a Salesforce object, that object displays as specified in the Salesforce console app settings. For example, if cases are set to open as a subtab of their parent accounts, and `openPrimaryTab()` is called on a case, the case opens as subtab on its parent account's primary tab.

If there's an error opening the tab, the error code is reported in the JavaScript console.

Syntax

```
sforce.console.openPrimaryTab(id:String, url:URL, active:Boolean,
(optional)tabLabel:String, (optional)callback:Function, (optional)name)
```

Arguments

Name	Type	Description
`id`	string	ID of the primary tab to override. Use `null` to create a new primary tab. If the ID corresponds to an existing primary tab, the existing tab is redirected to the given URL because the Salesforce console prevents duplicate tabs.
`url`	URL	URL of the opened primary tab. If the URL is to a Salesforce object, that object displays as specified in the Salesforce console app settings. For example, if cases are set to open as a subtab of their parent accounts, and `openPrimaryTab()` is called on a case, the case opens as subtab on its parent account's primary tab. Users can open an external URL if it's been added to the console's allowlist. Note: When using a relative URL, make sure that you include "/" at the beginning of your URL. When pointing to a Visualforce page, use "*/apex/*" at the beginning of your URL. Otherwise, your URL might not work as expected.
`active`	boolean	If `true`, the opened primary tab displays immediately. If `false`, the opened primary tab displays in the background and the current tab maintains focus.
`tabLabel`	string	Optional label of the opened primary tab. If a label isn't specified, `External Page` displays. Add labels as text; HTML isn't supported.
`callback`	function	JavaScript method called upon completion of the method.
`name`	string	Optional name of the opened primary tab. This argument is only available in API version 22.0 and later.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testOpenPrimaryTab();return false">
         Click here to open a new primary tab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testOpenPrimaryTab() {
            //Open a new primary tab with the salesforce.com home page in it
            sforce.console.openPrimaryTab(null, 'https://salesforce.com', false,
                'salesforce', openSuccess, 'salesforceTab');
        }

        var openSuccess = function openSuccess(result) {
            //Report whether opening the new tab was successful
```

```
            if (result.success == true) {
                alert('Primary tab successfully opened');
            } else {
                alert('Primary tab cannot be opened');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	`true` if the tab successfully opened; `false` if the tab didn't open.
`id`	string	ID of the primary tab. IDs are only valid during a user session; IDs become invalid when a user leaves the Salesforce console.

`openSubtab()`

Opens a new subtab (within a primary tab) that displays the content of a specified URL, which can be relative or absolute. You can also override an existing subtab. Use to open a new subtab on a primary tab via the primary tab's ID. This method is only available in API version 20.0 or later.

If there's an error opening the tab, the error code is reported in the JavaScript console.

Syntax

```
sforce.console.openSubtab(primaryTabId:String, url:URL, active:Boolean, tabLabel:String,
 id:String, (optional)callback:Function, (optional)name:String)
```

Arguments

Name	Type	Description
`primaryTabId`	string	ID of the primary tab in which the subtab opened.
`url`	URL	URL of the opened subtab. If the URL is to a Salesforce object, that object displays as specified in the Salesforce console app settings. For example, if cases are set to open as a primary tab, and `openSubtab()` is called on a case, the case opens as a primary tab. Users can open an external URL if it's been added to the console's allowlist.

Name	Type	Description
		Note: When using a relative URL, make sure that you include "/" at the beginning of your URL. When pointing to a Visualforce page, use "*/apex/*" at the beginning of your URL. Otherwise, your URL might not work as expected.
`active`	boolean	If `true`, the opened subtab displays immediately. If `false`, the opened subtab displays in the background and the current tab maintains focus.
`tabLabel`	string	Optional label of the opened subtab. If a label isn't specified, `External Page` displays. Add labels as text; HTML isn't supported.
`id`	string	ID of the subtab to override. Use `null` to create a new subtab.
`callback`	function	JavaScript method called upon completion of the method.
`name`	string	Optional name of the opened subtab. This argument is only available in API version 22.0 and later.

Sample Code–Visualforce

```
<apex:page standardController="Case">

    <A HREF="#" onClick="testOpenSubtab();return false">
        Click here to open a new subtab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testOpenSubtab() {
            //First find the ID of the primary tab to put the new subtab in
            sforce.console.getEnclosingPrimaryTabId(openSubtab);
        }

        var openSubtab = function openSubtab(result) {
            //Now that we have the primary tab ID, we can open a new subtab in it
            var primaryTabId = result.id;
            sforce.console.openSubtab(primaryTabId , 'https://salesforce.com', false,
                'salesforce', null, openSuccess, 'salesforceSubtab');
        };

        var openSuccess = function openSuccess(result) {
            //Report whether we succeeded in opening the subtab
            if (result.success == true) {
                alert('subtab successfully opened');
            } else {
                alert('subtab cannot be opened');
            }
        };
```

```
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in the Salesforce help.

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	`true` if the subtab successfully opened; `false` if the subtab didn't open.
`id`	string	ID of the subtab. IDs are only valid during a user session; IDs become invalid when the user leaves the Salesforce console.

`openSubtabByPrimaryTabName()`

Opens a new subtab (within a primary tab) that displays the content of a specified URL, which can be relative or absolute. You can also override an existing subtab. Use to open a new subtab on a primary tab via the primary tab's name. This method is only available in API version 22.0 or later.

If there's an error opening the tab, the error code is reported in the JavaScript console.

Syntax

```
sforce.console.openSubtabByPrimaryTabName(primaryTabName:String, url:URL, active:Boolean,
 tabLabel:String, id:String, (optional)callback:Function, (optional)name:String)
```

Arguments

Name	Type	Description
`primaryTabName`	string	Name of the primary tab in which the subtab opened.
`url`	URL	URL of the opened subtab. If the URL is to a Salesforce object, that object displays as specified in the Salesforce console app settings. For example, if cases are set to open as a primary tab, and `openSubtab()` is called on a case, the case opens as a primary tab. Users can open an external URL if it's been added to the console's allowlist.
`active`	boolean	If `true`, the opened subtab displays immediately. If `false`, the opened subtab displays in the background and the current tab maintains focus.
`tabLabel`	string	Optional label of the opened subtab. If a label isn't specified, `External Page` displays. Add labels as text; HTML isn't supported.

Name	Type	Description
`id`	string	ID of the subtab to override. Use `null` to create a new subtab.
`callback`	function	JavaScript method called upon completion of the method.
`name`	string	Optional name of the opened subtab. This argument is only available in API version 22.0 and later.

Sample Code–Visualforce

```
<apex:page standardController="Case">

    <A HREF="#" onClick="testOpenSubtab();return false">
        Click here to open a new subtab by primary tab name</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testOpenSubtabByPrimaryTabName() {
            //First open a primary tab by name
            sforce.console.openPrimaryTab(null, 'http://www.yahoo.com', true, 'Yahoo',
openSubtab, 'yahoo');
        }

        var openSubtab = function openSubtab(result) {
            //Open the subtab by the name specified in function
testOpenSubtabByPrimaryTabName()
            sforce.console.openSubtabByPrimaryTabName('yahoo', 'https://salesforce.com',
true,
                'salesforce', null, openSuccess);
        };

        var openSuccess = function openSuccess(result) {
            //Report whether we succeeded in opening the subtab
            if (result.success == true) {
                alert('subtab successfully opened');
            } else {
                alert('subtab cannot be opened');
            }
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	`true` if the subtab successfully opened; `false` if the subtab didn't open.
`id`	string	ID of the subtab. IDs are only valid during a user session; IDs become invalid when the user leaves the Salesforce console.

`refreshPrimaryTabById()`

Refreshes a primary tab specified by ID, including its subtabs. This method can't refresh subtabs with URLs to external pages or Visualforce pages. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.refreshPrimaryTabById(id:String, activate:Boolean,
(optional)callback:Function, (optional)fullRefresh:Boolean)
```

Arguments

Name	Type	Description
`id`	string	ID of the primary tab to refresh.
`activate`	boolean	If `true`, the refreshed primary tab displays immediately. If `false`, the refreshed primary tab displays in the background.
`callback`	function	JavaScript method that's called upon completion of the method.
`fullRefresh`	boolean	Enables a full refresh of the entire case feed.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testRefreshPrimaryTabById();return false">
         Click here to refresh a primary tab by id</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testRefreshPrimaryTabById() {
            //Get the value for 'scc-pt-0' from the openPrimaryTab method
            //This value is for example purposes only
            var primaryTabId = 'scc-pt-0';
            sforce.console.refreshPrimaryTabById(primaryTabId, true, refreshSuccess);
        }

        var refreshSuccess = function refreshSuccess(result) {
            //Report whether refreshing the primary tab was successful
            if (result.success == true) {
                alert('Primary tab refreshed successfully');
            } else {
```

```
                alert('Primary did not refresh');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the primary tab refreshed successfully; `false` if the primary tab didn't refresh.

`refreshPrimaryTabByName()`

Refreshes a primary tab specified by name, including its subtabs. This method can't refresh subtabs with URLs to external pages or Visualforce pages. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.refreshPrimaryTabByName(name:String, active:Boolean,
(optional)callback:Function), (optional)fullRefresh:Boolean)
```

Arguments

Name	Type	Description
`name`	string	Name of the primary tab to refresh.
`active`	boolean	If `true`, the refreshed primary tab displays immediately. If `false`, the refreshed primary tab displays in the background.
`callback`	function	JavaScript method that's called upon completion of the method.
`fullRefresh`	boolean	Enables a full refresh of the entire case feed.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testRefreshPrimaryTabByName();return false">
         Click here to refresh a primary tab by name</A>
```

```
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testRefreshPrimaryTabByName() {
            //Set the name of the tab by using the openPrimaryTab method
            //This value is for example purposes only
            var primaryTabName = 'myPrimaryTab';
            sforce.console.refreshPrimaryTabByName(primaryTabName, true, refreshSuccess);

        }

        var refreshSuccess = function refreshSuccess(result) {
            //Report whether refreshing the primary tab was successful
            if (result.success == true) {
                alert('Primary tab refreshed successfully');
            } else {
                alert('Primary tab did not refresh');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the primary tab refreshed successfully; `false` if the primary tab didn't refresh.

`refreshSubtabById()`

Refreshes a subtab with the last known URL with a specified ID. This method can't refresh a subtab if the last known URL is an external page or a Visualforce page. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.refreshSubtabById(id:String, activate:Boolean, (optional)callback:Function,
  (optional)fullRefresh:Boolean)
```

Arguments

Name	Type	Description
`id`	string	ID of the subtab to refresh.
`activate`	boolean	If `true`, the refreshed subtab displays immediately. If `false`, the refreshed subtab displays in the background.
`callback`	function	JavaScript method that's called upon completion of the method.
`fullRefresh`	boolean	Enables a full refresh of the entire case feed.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testRefreshSubtabById();return false">
         Click here to refresh a subtab by id</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testRefreshSubtabById() {
            //Set the name of the tab by using the openSubtab method
            //This value is for example purposes only
            var subtabId = 'scc-st-0';
            sforce.console.refreshSubtabById(subtabId, true, refreshSuccess);
        }

        var refreshSuccess = function refreshSuccess(result) {
            //Report whether refreshing the subtab was successful
            if (result.success == true) {
                alert('Subtab refreshed successfully');
            } else {
                alert('Subtab did not refresh');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the subtab refreshed successfully; `false` if the subtab didn't refresh.

refreshSubtabByNameAndPrimaryTabId()

Refreshes a subtab with the last known URL with the specified name and primary tab ID. This method can't refresh a subtab if the last known URL is an external page or a Visualforce page. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.refreshSubtabByNameAndPrimaryTabId(name:String, primaryTabId:String,
active:Boolean, (optional)callback:Function, (optional)fullRefresh:Boolean)
```

Arguments

Name	Type	Description
name	string	Name of the subtab to refresh.
primaryTabId	string	ID of the primary tab in which the subtab opened.
active	boolean	If true, the refreshed subtab displays immediately. If false, the refreshed subtab displays in the background.
callback	function	JavaScript method that's called upon completion of the method.
fullRefresh	boolean	Enables a full refresh of the entire case feed.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testRefreshSubtabByNameAndPrimaryTabId();return false">
         Click here to refresh a subtab by name and primary tab ID</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testRefreshSubtabByNameAndPrimaryTabId() {
            //Get the value for 'mySubtab' and 'scc-pt-0' from the openSubtab method
            //These values are for example purposes only
            var subtabName = 'mySubtab';
            var primaryTabId = 'scc-pt-0';
            sforce.console.refreshSubtabByNameAndPrimaryTabId(subtabName, primaryTabId,
true, refreshSuccess);
        }

        var refreshSuccess = function refreshSuccess(result) {
            //Report whether refreshing the subtab was successful
            if (result.success == true) {
                alert('Subtab refreshed successfully');
            } else {
                alert('Subtab did not refresh');
            }
        };
```

```
  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the subtab refreshed successfully; `false` if the subtab didn't refresh.

`refreshSubtabByNameAndPrimaryTabName()`

Refreshes a subtab with the last known URL with the specified name and primary tab name. This method can't refresh a subtab if the last known URL is an external page or a Visualforce page. This method is only available in API version 22.0 or later.

Syntax

```
sforce.console.refreshSubtabByNameAndPrimaryTabName(name:String, primaryTabName:String,
active:Boolean, (optional)callback:Function, (optional)fullRefresh:Boolean)
```

Arguments

Name	Type	Description
`name`	string	Name of the subtab to refresh.
`primaryTabName`	string	Name of the primary tab in which the subtab opened.
`active`	boolean	If `true`, the refreshed subtab displays immediately. If `false`, the refreshed subtab displays in the background.
`callback`	function	JavaScript method that's called upon completion of the method.
`fullRefresh`	boolean	Enables a full refresh of the entire case feed.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testRefreshSubtabByNameAndPrimaryTabName();return false">
         Click here to refresh a subtab by name and primary tab name</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
```

```
        function testRefreshSubtabByNameAndPrimaryTabName() {
            //Get the value for 'mySubtab' and 'myPrimaryTab' from the openSubtab method
            //These values are for example purposes only
            var subtabName = 'mySubtab';
            var primaryTabName = 'myPrimaryTab';
          sforce.console.refreshSubtabByNameAndPrimaryTabName(subtabName, primaryTabName,
 true, refreshSuccess);
        }

        var refreshSuccess = function refreshSuccess(result) {
            //Report whether refreshing the subtab was successful
            if (result.success == true) {
                alert('Subtab successfully refreshed');
            } else {
                alert('Subtab did not refresh');
            }
        };

  </script>

</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the subtab refreshed successfully; `false` if the subtab didn't refresh.

`reopenLastClosedTab()`

Reopens the last closed primary tab, and any of its subtabs that were open, the moment it was closed. This method is only available in API version 35.0 or later.

Syntax

```
sforce.console.reopenLastClosedTab()
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var  = callback = function (result) {
            if (result.success) {
                alert('Last tab was re-opened!');
            } else {
                alert('No tab was re-opened.');
            }
        };

        function reopenLastClosedTabTest() {
               sforce.console.reopenLastClosedTab(callback);
        }

  </script>
  <A HREF="#" onClick="reopenLastClosedTabTest(); return false">Re-open Last Closed Tab</A>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the tab was reopened, `false` otherwise.

`resetSessionTimeOut()`

Resets a session timeout for a console app. This method ensures that users can continue working on Visualforce pages without being prompted to log back in to the console when they return to a console tab or console component. This method is only available in API version 24.0 or later.

For more information, see Modify Session Security Settings in Salesforce Help.

Syntax

```
sforce.console.resetSessionTimeOut()
```

Arguments

None

Sample Code–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testResetSessionTimeOut();">
```

```
            Click here to reset session timeout</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testResetSessionTimeOut() {
            sforce.console.resetSessionTimeOut();
        };
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

None

setTabUnsavedChanges()

Sets the unsaved changes icon (*) on subtabs to indicate unsaved data. This method is only available in API version 23.0 or later.

Syntax

```
sforce.console.setTabUnsavedChanges(unsaved:Boolean, callback:Function,
(optional)subtabId:String)
```

Arguments

Name	Type	Description
unsaved	boolean	If true, the tab is marked as having unsaved changes.
callback	function	JavaScript method that's called upon completion of the method.
subtabId	string	The ID of the subtab that is marked as having unsaved changes. This argument is only available in API version 25.0 or later.

Sample Code API Version 23.0 or Later–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testSetTabUnsavedChanges();return false">
            Click here to indicate this tab has unsaved changes</A>

<apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetTabUnsavedChanges() {
            sforce.console.setTabUnsavedChanges(true, displayResult);
        };
        function displayResult(result) {
             if (result.success) {
```

```
                alert('Tab status has been successfully updated');
            } else {
                alert('Tab status couldn't be updated');
            }
        }

    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in the Salesforce help.

Response

This method returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if update was successful; `false` if update wasn't successful.

Sample Code API Version 25.0 or Later–Visualforce

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testSetTabUnsavedChanges();return false">
           Click here to indicate this tab has unsaved changes</A>

    <apex:includeScript value="/support/console/25.0/integration.js"/>
    <script type="text/javascript">
         function testSetTabUnsavedChanges() {
            sforce.console.getFocusedSubtabId(setTabDirty);
        };
         function setTabDirty(result) {
            sforce.console.setTabUnsavedChanges(true, displayResult, result.id);
        };
         function displayResult(result) {
            if (result.success) {
                alert('Tab status has been successfully updated');
             } else {
                alert('Tab status couldn't be updated');
             }
        };
    </script>
</apex:page>
```

Note: This example is only set to run if the Visualforce page is inside an application-level custom component. For more information, see Methods for Application-Level Custom Console Components on page 250.

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if returning the focused subtab ID was successful; `false` if if returning the focused subtab ID wasn't successful.

`setTabIcon()`

Sets an icon on the specified tab. If a tab is not specified, the icon is set on the enclosing tab. Use this method to customize a tab's icon. This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.setTabIcon(iconUrl:String, tabID:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`iconUrl`	string	A URL pointing to an image, which is used as the tab's icon. If null or undefined, the tab's default icon is used.
`tabId`	string	The ID of the tab on which to set the icon. If null or undefined, the enclosing tab's ID is used.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testSetTabIcon();return false">
         Click here to change the enclosing tab's icon</A> <BR/>
    <A HREF="#" onClick="testResetTabIcon(); return false;">
         Click here to reset the enclosing tab's icon</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function checkResult(result) {
            if (result.success) {
               alert('Tab icon set successfully!');
            } else {
               alert('Tab icon cannot be set!');
            }
    }
        function testSetTabIcon() {
            sforce.console.setTabIcon('http://host/path/to/your/icon.png', null,
checkResult);
    }
        function testResetTabIcon() {
            sforce.console.setTabIcon(null, null, checkResult);
    }
```

```
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if setting the tab's icon was successful, `false` if setting the tab's icon wasn't successful.

Note: If this method is called without passing in a tab ID, the tab in which the Visualforce page is enclosed is used. If there isn't an enclosing tab, the error message `Cannot get a workspace or view tab from the given ID` displays in the browser's developer console.

`setTabLink()`

Sets a console tab's URL attribute to the location of the tab's content. Use this method to generate secure console URLs when users navigate to tabs displaying content outside of the Salesforce domain.This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.setTabLink((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page standardController="Account">
    <apex: detail />
        <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        window.onload = function() {
            sforce.console.setTabLink();
             };
  </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	`true` if the link was set successfully, `false` if setting was unsuccessful.
`callback`	function	JavaScript method that's called upon completion of the method.

`setTabStyle()`

Sets a cascading style sheet (CSS) on the specified tab. If a tab is not specified, the CSS is set on the enclosing tab. Use this method to customize a tab's look and feel. This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.setTabStyle(style:String, tabId:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`style`	string	A CSS specification string used to style the tab. If null or undefined, the tab's default style is used.
`tabId`	string	The ID of the tab on which to set the style. If null or undefined, the enclosing tab's ID is used.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testSetTabStyle();return false">
         Click here to change the enclosing tab's background color to red</A> <BR/>
    <A HREF="#" onClick="testResetTabStyle(); return false;">
         Click here to reset the enclosing tab's style</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function checkResult(result) {
            if (result.success) {
               alert('Tab style set successfully!');
            } else {
               alert('Tab style cannot be set!');
            }
    }
        function testSetTabStyle() {
            sforce.console.setTabStyle('background:red;', null, checkResult);
```

```
    }
        function testResetTabStyle() {
            sforce.console.setTabStyle(null, null, checkResult);
    }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if setting the tab's style was successful, `false` if setting the tab's style wasn't successful.

Note: If this method is called without passing in a tab ID, the tab in which the Visualforce page is enclosed is used. If there isn't an enclosing tab, the error message `Cannot get a workspace or view tab from the given ID` displays in the browser's developer console.

`setTabTextStyle()`

Sets a cascading style sheet (CSS) on a specified tab's text. If a tab is not specified, the CSS is set on the enclosing tab's text. Use this method to customize a tab's text style. This method is only available in API version 28.0 or later.

Syntax

```
sforce.console. setTabTextStyle(style:String, tabID:String, (optional)callback:Function))
```

Arguments

Name	Type	Description
`style`	string	A CSS specification string used to set a tab's text style. If null or undefined, the tab's default text style is used.
`tabId`	string	The ID of the tab on which to set the text style. If null or undefined, the enclosing tab's ID is used.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testSetTabTextStyle();return false">
         Click here to change the enclosing tab's text style</A> <BR/>
    <A HREF="#" onClick="testResetTabTextStyle(); return false;">
```

```
        Click here to reset the enclosing tab's text style</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function checkResult(result) {
            if (result.success) {
               alert('Tab text style set successfully!');
            } else {
               alert('Tab text style cannot be set!');
            }
    }
        function testSetTabTextStyle() {
            sforce.console.setTabTextStyle('color:blue;font-style:italic;', null,
checkResult);
    }
        function testResetTabTextStyle() {
            sforce.console.setTabTextStyle(null, null, checkResult);
    }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if setting the tab's text style was successful, `false` if setting the tab's text style wasn't successful.

Note: If this method is called without passing in a tab ID, the tab in which the Visualforce page is enclosed is used. If there isn't an enclosing tab, the error message `Cannot get a workspace or view tab from the given ID` displays in the browser's developer console.

`setTabTitle()`

Sets the title of a primary tab or subtab. This method is only available in API version 20.0 or later.

Syntax

```
sforce.console.setTabTitle(tabTitle:String, (optional)tabID:String)
```

Arguments

Name	Type	Description
`tabTitle`	string	Title of a primary tab or subtab.

Name	Type	Description
tabId	string	The ID of the tab in which to set the title. This argument is only available in API version 25.0 or later.

Sample Code–Visualforce API Version 20.0 or Later

```
<apex:page standardController="Case">
    <A HREF="#" onClick="testSetTabTitle();return false">
         Click here to change this tab's title</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetTabTitle() {
            //Set the current tab's title
            sforce.console.setTabTitle('My New Title');
        }
    </script>
</apex:page>
```

Note: To see this example in action, click the custom link on a case. For more information, see Define Custom Buttons and Links in Salesforce Help.

Response

None

Sample Code–Visualforce API Version 25.0 or Later

```
<apex:page>
    <A HREF="#" onClick="testSetTabTitle();return false">
         Click here to change the title of the focused primary tab</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetTabTitle() {
            sforce.console.getFocusedPrimaryTabId(function(result) {
               sforce.console.setTabTitle('My New Title', result.id);
            });
 }
    </script>
</apex:page>
```

Note: This example is only set to run if the Visualforce page is inside an application-level custom component. For more information, see Methods for Application-Level Custom Console Components on page 250.

Response

None

Methods for Navigation Tabs

A Salesforce console displays a navigation tab from which users can select objects to view lists or home pages. Administrators choose the objects that users can access from a navigation tab.

IN THIS SECTION:

`focusNavigationTab()`

Focuses the browser on the navigation tab. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.focusNavigationTab((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var callback = function (result) {}
            if(result.success){
            alert('success');
        }
        else{
        alert('Something is wrong.');
        }
    };
        sforce.console.focusNavigationTab(callback);
```

```
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if returning the object IDs was successful; `false` otherwise.

`getNavigationTabs()`

Returns all of the objects in the navigation tab. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.getNavigationTabs((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var callback = function (result) {
           var id;
              if (result.success) {
                var tempItem = JSON.parse(result.items);
                for (var i = 0, len = tempItem.length; i < len; i++) {

alert('Label:'+tempItem[i].label+'listViewURl:'+tempItem[i].listViewUrl+'navTabid:'
           +tempItem[i].navigationTabId+'Selected ' +tempItem[i].selected);
           }
              } else {
                alert('something is wrong!');
                }
        };
        sforce.console.getNavigationTabs(callback);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`menuItems`	object	The IDs of objects in the navigation tab.
`success`	boolean	`true` if returning the IDs of objects in the navigation tab was successful, `false` otherwise.

`getSelectedNavigationTab()`

Returns the selected object in the navigation tab. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.getSelectedNavigationTab((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var callback = function (result) {}
            if (result.success) {
             alert('the navigation tab id is ' + result.navigationTabId + ' and navigation
 url is ' + result.listViewUrl);
                           } else {
               alert('something is wrong!');
            }
        };
        sforce.console.getSelectedNavigationTab(callback);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`navigationTabId`	string	The object ID of the selected object.
`listViewUrl`	object	The list view URL of the selected object.
`label`	object	The label of the selected object.
`selected`	boolean	`true` if returning the selected field of the object was successful, `false` otherwise.
`success`	boolean	`true` if returning the object IDs was successful, `false` otherwise.

`refreshNavigationTab()`

Refreshes the selected navigation tab. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.refreshNavigationTab((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var callback = function (result) {}
            if(result.success){
              alert('success');
            }
            else{
              alert('Something is wrong.');
            }
        };
        sforce.console.refreshNavigationTab(callback);
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if refreshing the navigation tab was successful, `false` otherwise.

`setSelectedNavigationTab()`

Sets the navigation tab with a specific ID or URL. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.setSelectedNavigationTab((optional)callback, navigatorTabId:(optional)string,
 url:(optional)string)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.
`navigatorTabId`	string	The ID of the navigation tab to be selected.
`url`	string	The URL of the navigation tab to be selected.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var callback = function (result) {}
            if (result.success) {
              alert('Successful');
            } else {
              alert('something is wrong!');
            }
        };
        sforce.console.setSelectedNavigationTab(callback,'nav-tab-4');
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if setting the navigation tab with a specific ID or URL was successful, `false` otherwise.

Methods for Computer-Telephony Integration (CTI)

Salesforce Call Center seamlessly integrates Salesforce with Computer-Telephony Integration systems. Developers create a CTI system with and console users access telephony features through a softphone, which is a call-control tool that appears in the footer of a console.

IN THIS SECTION:

`fireOnCallBegin()`

Fires an event that notifies a call has begun. Use to get information or send information between an interaction log and a custom console component. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.cti.fireOnCallBegin( callObjectId:String, callType:String, callLabel:String,
 (optional)callback:Function )
```

Arguments

Name	Type	Description
`callObjectId`	string	The object ID of the call.
`callType`	string	String that specifies the call type, which must be `internal`, `inbound` or `outbound`.
`callLabel`	string	String that specifies a call as it appears in the **Attach Call** drop-down button. For example, `Call Label — Inbound Call 12:52:31 PM`.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testFireOnCallBegin();return false">
             Click here to start a call</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        function testFireOnCallBegin() {
            sforce.console.cti.fireOnCallBegin('call.794937' , 'outbound' , 'label 1');
        }

    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if firing the event is successful, `false` otherwise.

`fireOnCallEnd()`

Fires an event that notifies a call has ended. Use to get information or send information between an interaction log and a custom console component. This method executes when `fireOnCallBegin()` is called first. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.cti.fireOnCallEnd( callObjectId:String, callDuration:Number,
callDisposition:String, (optional)callback:Function )
```

Arguments

Name	Type	Description
`callObjectId`	string	The object ID of the call.
`callDuration`	number	Number specifying the duration of the call.
`callDisposition`	string	String representing the call's disposition, such as call successful, left voicemail, or disconnected.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testFireOnCallEnd();return false">
             Click here to end a call</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        function testFireOnCallEnd() {
            //Here, 'call.1' refers to a call that is in progress.
            sforce.console.cti.fireOnCallEnd('call.1', 60, 'Set Appointment');
        }

    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if firing the event is successful, `false` otherwise.

`fireOnCallLogSaved()`

Calls the `eventHandler` function registered with `onCallLogSaved()`. Use to get information or send information between an interaction log and a custom console component.. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.cti.fireOnCallLogSaved( id:String, (optional)callback:Function )
```

Arguments

Name	Type	Description
`id`	string	The object ID of the saved call log.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var MyCallback = function (result) {
             alert('fireOnCallLogSaved was thrown: ' + result.success);
        };

        function testFireOnCallLogSaved() {
            // Simulates that a call log was saved by passing the task object Id as input.

             sforce.console.cti.fireOnCallLogSaved('00Txx000003qf8u', myCallback);
        }

        var callback = function (result) {
            alert('Call Log was saved! Object Id saved is : ' + result.id);
        };

        sforce.console.cti.onCallLogSaved(callback);
    </script>
    <a href="#" onClick="testFireOnCallLogSaved();return false">
        Test fireOnCallLogSaved API!</a>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if firing the event is successful, `false` otherwise.

`getCallAttachedData()`

Returns the attached data of a call represented by the call object ID or null if there isn't an active call. The data is returned in JSON format. This method is for computer-telephony integration (CTI); it's only available in API version 24.0 or later.

Syntax

```
sforce.console.cti.getCallAttachedData( callObjectId, getCallType, (optional)
callback:Function )
```

Arguments

Name	Type	Description
`callObjectId`	string	The call object ID of the call that retrieves the attached data.
`getCallType`	boolean	`true` if the type of call is returned as either 'INTERNAL,' 'INBOUND,' or 'OUTBOUND'; `false` otherwise. This field is only available in API version 29.0 or later.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

     /* Note: Open CTI needs to set call type to before getting it, and call type is
INTERNAL/INBOUND/OUTBOUND.
      */

           var callback2 = function (result) {
              alert('Call attached data  is ' + result.data + '\n Call Type is ' +
result.type);

           };

           /* Retrieving call ID of first call that came in and
            * calling getCallAttachedData() to retrieve call data.
            */
           var callback1 = function (result) {
                if (result.ids && result.ids.length > 0) {
                  sforce.console.cti.getCallAttachedData(result.ids[0], callback2,
{getCallType:true});
                }
           };

          //Note that we are using the CTI submodule here
          function testGetCallAttachedData() {
            sforce.console.cti.getCallObjectIds(callback1);
         };
```

```
    </script>
  <h1>CTI</h1>
   <button onclick="testGetCallAttachedData()">getAttachedData</button>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`data`	string	The attached data of a call in JSON format.
`success`	boolean	`true` if returning the attached data was successful; `false` if returning the attached data wasn't successful.
`type`	string	The type of call. Possible values are 'INTERNAL', 'INBOUND', and 'OUTBOUND'.

`getCallObjectIds()`

Returns any active call object IDs in the order in which they arrived or null if there aren't any active calls. This method is for computer-telephony integration (CTI); it's only available in API version 24.0 or later.

Syntax

```
sforce.console.cti.getCallObjectIds( (optional) callback:Function )
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

          var callback = function (result) {
              alert('Active call object ids: ' + result.ids);
          };

          //Note that we are using the CTI submodule here
          sforce.console.cti.getCallObjectIds(callback);
     </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`ids`	string	The call object IDs of active calls or null if no call is active.
`success`	boolean	`true` if returning the active call object IDs was successful; `false` if returning the active call object IDs wasn't successful.

`onCallBegin()`

Registers a function that is called when a call begins (comes in). This method is for computer-telephony integration (CTI); it's only available in API version 24.0 or later.

Syntax

```
sforce.console.cti.onCallBegin( eventHandler:Function )
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when a call begins.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

           var callback = function (result) {
              alert('Call ' + result.id + 'Just came in!');
           };

          //Note that we are using the CTI submodule here
           sforce.console.cti.onCallBegin(callback);
     </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`id`	string	The call object ID of the call which has begun.

onCallEnd()

Registers a function that is called when a call ends. This method is for computer-telephony integration (CTI); it's only available in API version 24.0 or later.

Syntax

```
sforce.console.cti.onCallEnd( eventHandler:Function )
```

Arguments

Name	Type	Description
eventHandler	function	JavaScript method called when a call ends.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

           var callback = function (result) {
              var str = 'Call ' + result.id + ' ended! ';
              str += 'Call duration is ' + result.duration + '. ';
              str += 'Call disposition is ' + result.disposition;
              alert(str);
           };

          //Note that we are using the CTI submodule here
           sforce.console.cti.onCallEnd(callback);
     </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
id	string	The call object ID of the call which has ended.
duration	string	The duration of the call.
disposition	string	The disposition of the call.

onCallLogSaved()

Registers a function that is fired when an interaction log saves a call log. Use to get information or send information between an interaction log and a custom console component. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.cti.onCallLogSaved( eventHandler:Function )
```

Arguments

Name	Type	Description
`eventHandler`	function	For a standard interaction log, `eventHandler` is a function that is executed when a call log is saved. For a custom interaction log, `eventHandler` is a function that is executed when the `fireOnCallLogSaved` API is called in your Visualforce page.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var callback = function (result) {
            alert('Call Log was saved! Object Id saved is : ' + result.id);
        };

        sforce.console.cti.onCallLogSaved(callback);

  </script>
    <p>Registered onCallLogSaved listener...</p>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`id`	string	Call log object ID that was saved.

`onSendCTIMessage()`

Registers a function that is fired when a message is sent with the `sendCTIMessage()`. Use to get information or send information between an interaction log and a custom console component. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.cti.onSendCTIMessage( eventHandler:Function )
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when a message is sent with the `sendCTIMessage()` method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

           var callback = function (result) {
               alert('sendCTIMessage API sent the following message: ' + result.message);

           };

           sforce.console.cti.onSendCTIMessage(callback);

           function sendCTIMessage() {
               sforce.console.cti.sendCTIMessage('sending a message to CTI');
           }
     </script>
     <a href="#" onClick="sendCTIMessage();return false">
                 Send a message to see your listener receiving it!</a>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`message`	string	The message that was sent with the `sendCTIMessage()` method.

`sendCTIMessage()`

Sends a message to the CTI adapter or Open CTI. This method is for computer-telephony integration (CTI); it's only available in API version 24.0 or later.

Syntax

```
sforce.console.cti.sendCTIMessage( msg, (optional) callback:Function )
```

Arguments

Name	Type	Description
`msg`	string	Message to send to the adapter.
`callback`	function	JavaScript method called when the message is sent.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

           var callback = function (result) {
              if (result.success) {
                 alert('CTI message was sent successfully!');
              } else {
                 alert('CTI message was not sent successfully.');
               }
           };

          //Note that we are using the CTI submodule here
           sforce.console.cti.sendCTIMessage('/ANSWER?LINE_NUMBER=1', callback);
     </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if sending the message was successful; `false` if sending the message wasn't successful.

`setCallAttachedData()`

Sets the call data associated with a call object ID. Use to get information or send information between an interaction log and a custom console component.This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.cti.setCallAttachedData( callObjectId:String, callData:JSON string
callType:String, (optional)callback:Functional)
```

Arguments

Name	Type	Description
`callObjectId`	string	The object ID of the call.
`callData`	string	JSON string that specifies the data to attach to the call.
`callType`	string	String that specifies the call type, such as `internal`, `inbound`, or `outbound`.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testSetCallAttachedData();return false">
          Click here to set call attached data </A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

           function testSetCallAttachedData() {
              //callData must be a JSON string. We assume that your browser has
              //access to a JSON library.
              var callData = JSON.stringify({"ANI":"4155551212", "DNIS":"8005551212"});

              //Set the call attached data associated to call id 'call.1'
              sforce.console.cti.setCallAttachedData('call.1', callData, 'outbound');
           }
     </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the event firing was successful; `false` otherwise.

`setCallObjectIds()`

Sets call object IDs, in ascending order of arrival. This method is only available in API version 31.0 or later.

Syntax

```
sforce.console.cti.setCallObjectIds( callObjectIds:Array, callback:Function )
```

Arguments

Name	Type	Description
`callObjectId`	array	An array of string IDs specifying the calls to set.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="testSetCallObjectIds();return false">
             Click here to set call object Ids</A>

     <apex:includeScript value="/support/console/63.0/integration.js"/>
     <script type="text/javascript">

          function checkResult(result) {
              if (result.success) {
                    alert('Call object ids set successfully!');
              } else {
                    alert('Call object ids cannot be set!');
              }
          }

          function testSetCallObjectIds() {
              sforce.console.cti.setCallObjectIds(['call.1', 'call.2', 'call.3'],
checkResult);
          }
     </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if setting the call IDs was successful; `false` otherwise.

Methods for Application-Level Custom Console Components

Custom console components let you customize, extend, or integrate the footer, sidebars, highlights panels, and interaction logs of a Salesforce console using Visualforce, canvas apps, lookup fields, or related lists. Administrators can add components to either:

- Page layouts to display content on specific pages
- Salesforce console apps to display content across all pages and tabs

IN THIS SECTION:

addToBrowserTitleQueue()

Adds a browser tab title to a list of titles, which rotates every three seconds. This method is only available in API version 28.0 or later.

blinkCustomConsoleComponentButtonText()

Blinks a button's text on an application-level custom console component that's on a page. This method is only available in API version 25.0 or later. This method isn't supported in Lightning Console.

isCustomConsoleComponentPoppedOut()

Determines if a custom console component is popped out from a browser. To use this method, multi-monitor components must be turned on. This method is only available in API version 30.0 or later.

isCustomConsoleComponentWindowHidden()

Determines if the application-level custom console component window is hidden. This method is available in API versions 25.0 through 31.0.

isCustomConsoleComponentHidden()

Determines if the application-level custom console component window is hidden. This method is available in API version 32.0 and later. In API version 31.0 and earlier, this method was called `isCustomConsoleComponentWindowHidden()`.

isInCustomConsoleComponent()

Determines if the page is in an application-level custom console component. This method is only available in API version 25.0 or later.

onCustomConsoleComponentButtonClicked()

Registers a function to call when a button is clicked on an application-level custom console component. This method is only available in API version 25.0 or later.

removeFromBrowserTitleQueue()

Removes a browser tab title from the list of titles, which rotates every three seconds. This method is only available in API version 28.0 or later.

runSelectedMacro()

Executes the selected macro in the macro widget. This method is only available in API version 36.0 or later. This method isn't supported in Lightning Console.

scrollCustomConsoleComponentButtonText()

Scrolls a button's text on an application-level custom console component that's on a page. This method is only available in API version 25.0 or later. This method isn't supported in Lightning Console.

selectMacro()

Selects and displays a specific macro in the macro widget. This method is only available in API version 36.0 or later. This method isn't supported in Lightning Console.

setCustomConsoleComponentButtonIconUrl()

Sets the button icon URL of an application-level custom console component that's on a page. This method is only available in API version 25.0 or later.

setCustomConsoleComponentButtonStyle()

Sets the style of a button used to launch an application-level custom console component that's on a page. This method is only available in API version 25.0 or later. This method isn't supported in Lightning Console.

setCustomConsoleComponentButtonText()

Sets the text on a button used to launch an application-level custom console component that's on a page. This method is only available in API version 25.0 or later.

setCustomConsoleComponentHeight()

Sets the window height of an application-level custom console component that's on a page. This method is available in API version 32.0 or later.

setCustomConsoleComponentVisible()

Sets the window visibility of an application-level custom console component that's on a page. This method is available in API version 32.0 and later. In API version 31.0 and earlier, this method was called `setCustomConsoleComponentWindowVisible()`.

setCustomConsoleComponentWidth()

Sets the window width of an application-level custom console component that's on a page. This method is available in API version 32.0 or later.

setCustomConsoleComponentPopoutable()

Sets a custom console component to be popped out or popped into a browser. To use this method, multi-monitor components must be turned on. This method is only available in API version 30.0 or later.

setCustomConsoleComponentWindowVisible()

Sets the window visibility of an application-level custom console component that's on a page. This method is available in API versions 25.0 through 31.0. This method isn't supported in Lightning Console.

setSidebarVisible()

Shows or hides a console sidebar based on `tabId` and region. This method is available in API version 33.0 or later. This method isn't supported in Lightning Console.

addToBrowserTitleQueue()

Adds a browser tab title to a list of titles, which rotates every three seconds. This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.addToBrowserTitleQueue( title:String, callback:Function )
```

Arguments

Name	Type	Description
`title`	string	Browser tab title that is displayed.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <A HREF="#" onClick="testAddToBrowserTitleQueue();return false">
            Click here to enqueue a browser title</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
                function testAddToBrowserTitleQueue() {
             var title = 'TestTitle';
          sforce.console.addToBrowserTitleQueue(title);
```

```
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	If `true`, the title was successfully added to the browser title queue. If `false`, the title wasn't added to the browser title queue.
`callback`	function	JavaScript method that's called upon completion of the method.

`blinkCustomConsoleComponentButtonText()`

Blinks a button's text on an application-level custom console component that's on a page. This method is only available in API version 25.0 or later. This method isn't supported in Lightning Console.

Syntax

```
sforce.console.blinkCustomConsoleComponentButtonText(alternateText:String, interval:number,
 (optional)callback:Function)
```

Arguments

Name	Type	Description
`alternateText`	string	The alternate text to display when the button text blinks.
`interval`	number	Controls how often the text blinks in milliseconds.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testBlinkCustomConsoleComponentButtonText();return false">
         Click here to blink the button text on a custom console component</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testBlinkCustomConsoleComponentButtonText() {
        //Blink the custom console component button text
            sforce.console.blinkCustomConsoleComponentButtonText('Hello World', 10,
function(result){
             if (result.success) {
```

```
                alert('The text blinking starts!');
              } else {
                alert('Could not initiate the text blinking!');
              }
            });
          }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if blinking, the button text was successful; `false` if blinking the button text wasn't successful.

`isCustomConsoleComponentPoppedOut()`

Determines if a custom console component is popped out from a browser. To use this method, multi-monitor components must be turned on. This method is only available in API version 30.0 or later.

Syntax

```
sforce.console.isCustomConsoleComponentPoppedOut (callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="checkPoppedOut(); return false;">
         Is this component popped out?</A> <BR/>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function checkResult(result) {
          if (result.success) {
            alert('Is this component popped out: ' + result.poppedOut);
          } else {
            alert('Error invoking this method. Check the browser developer console for
more information.');
```

```
        }
      }
      function checkPoppedOut() {
        sforce.console.isCustomConsoleComponentPoppedOut(checkResult);
      }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if returning the component's pop out status was successful; `false` otherwise.
`poppedOut`	boolean	`true` if the component is popped out; `false` otherwise.

`isCustomConsoleComponentWindowHidden()`

Determines if the application-level custom console component window is hidden. This method is available in API versions 25.0 through 31.0.

Note: If this method is called from a popped out component in a Salesforce console where multi-montior components is turned on, nothing will happen. Starting in API version 32.0, this method is no longer available and has been replaced by `isCustomConsoleComponentHidden()`. For more information, see "`isCustomConsoleComponentHidden()`."

Syntax

```
sforce.console.isCustomConsoleComponentWindowHidden((optional) callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

     <A HREF="#" onClick="testIsCustomConsoleComponentWindowHidden();return false">
         Click here to check if the custom console component window is hidden</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testIsCustomConsoleComponentWindowHidden() {
            sforce.console.isCustomConsoleComponentWindowHidden(checkWindowVisibility);
```

```
        }

        var checkWindowVisibility = function checkWindowVisibility(result) {
            //Display the window visibility
            if (result.success) {
              alert('Is window hidden: ' + result.hidden);
            } else {
              alert('Error!');
            }
        }
    </script>

</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`hidden`	boolean	`true` if the custom console component window is hidden; `false` if the custom console component window is visible.
`success`	boolean	`true` if returning the custom console component window visibility was successful; `false` if returning the custom console component window visibility wasn't successful.

`isCustomConsoleComponentHidden()`

Determines if the application-level custom console component window is hidden. This method is available in API version 32.0 and later. In API version 31.0 and earlier, this method was called `isCustomConsoleComponentWindowHidden()`.

Syntax

```
sforce.console.isCustomConsoleComponentHidden((optional) callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

     <A HREF="#" onClick="testIsCustomConsoleComponentHidden();return false">
         Click here to check if the custom console component window is hidden</A>
```

```
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testIsCustomConsoleComponentHidden() {
            sforce.console.isCustomConsoleComponentHidden(checkWindowVisibility);
        }

        var checkWindowVisibility = function checkWindowVisibility(result) {
            //Display the window visibility
            if (result.success) {
              alert('Is window hidden: ' + result.hidden);
            } else {
              alert('Error!');
            }
        }
  </script>

</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`hidden`	boolean	`true` if the custom console component window is hidden; `false` if the custom console component window is visible.
`success`	boolean	`true` if the `isCustomConsoleComponentHidden()` call was successful; `false` if the `isCustomConsoleComponentHidden()` call wasn't successful.

`isInCustomConsoleComponent()`

Determines if the page is in an application-level custom console component. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.isInCustomConsoleComponent((optional) callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

     <A HREF="#" onClick="testIsInCustomConsoleComponent();return false">
         Click here to check if the page is in an app-level custom console component</A>


    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testIsInCustomConsoleComponent() {
            sforce.console.isInCustomConsoleComponent(checkInComponent);
        }

        var checkInComponent = function checkInComponent(result) {
            //Check if in component
            alert('Is in custom console component: ' + result.inCustomConsoleComponent);
        };

  </script>

</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`inCustomConsoleComponent`	boolean	`true` if the page is in a custom console component; `false` if the page isn't in a custom console component.
`success`	boolean	`true` if returning the page status was successful; `false` if returning the page status wasn't successful.

`onCustomConsoleComponentButtonClicked()`

Registers a function to call when a button is clicked on an application-level custom console component. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.onCustomConsoleComponentButtonClicked(eventHandler:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called when a button is clicked on a custom console component.

Sample Code–Visualforce

```
<apex:page>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
             var eventHandler = function (result) {
                 alert('The Custom Console Component button is clicked. The component ID
is: ' + result.id +
                 ' and the component window is: ' + (result.windowHidden ? 'hidden' :
'visible'));
        };

        sforce.console.onCustomConsoleComponentButtonClicked(eventHandler);
    </script>

</apex:page>
```

Event Handler Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`id`	string	The ID of the custom console component which includes the page.
`windowHidden`	boolean	`true` if the custom console component window is hidden after the button is clicked; `false` if the custom console component window is visible after the button is clicked.

`removeFromBrowserTitleQueue()`

Removes a browser tab title from the list of titles, which rotates every three seconds. This method is only available in API version 28.0 or later.

Syntax

```
sforce.console.removeFromBrowserTitleQueue( title:String, callback:Function )
```

Arguments

Name	Type	Description
`title`	string	Browser tab title to remove.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        <A HREF="#" onClick="testAddToBrowserTitleQueue();return false"> {
            Click here to enqueue a browser title</A>

        <A HREF="#" onClick="testRemoveFromBrowserTitleQueue();return false">
            Click here to remove browser title</A>

        var title = 'TestTitle';
        function testAddToBrowserTitleQueue() {
            sforce.console.addToBrowserTitleQueue(title);
        }
        function testRemoveFromBrowserTitleQueue() {
            sforce.console.removeFromBrowserTitleQueue(title);
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`success`	boolean	If `true`, the title was successfully removed from the browser title queue. If `false`, the title wasn't removed from the browser title queue.
`callback`	function	JavaScript method that's called upon completion of the method.

`runSelectedMacro()`

Executes the selected macro in the macro widget. This method is only available in API version 36.0 or later. This method isn't supported in Lightning Console.

Syntax

```
sforce.console.runSelectedMacro ((optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that is called when the method is completed

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="executeInWidget();return false">Click here to run a macro</A>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function executeInWidget() {
           sforce.console.runSelectedMacro();
        }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`cause`	string	Explanation of function failure, if applicable
`success`	boolean	`true` if running the macro was successful; `false` otherwise

`scrollCustomConsoleComponentButtonText()`

Scrolls a button's text on an application-level custom console component that's on a page. This method is only available in API version 25.0 or later. This method isn't supported in Lightning Console.

Syntax

```
sforce.console.scrollCustomConsoleComponentButtonText(interval:number, pixelsToScroll:number,
 isLeftScrolling:boolean, (optional)callback:Function)
```

Arguments

Name	Type	Description
`interval`	number	Controls how often the button text is scrolled in milliseconds.
`pixelsToScroll`	number	Controls how many pixels the button text scrolls.
`isLeftScrolling`	boolean	If `true`, the text scrolls left. If `false`, the text scrolls right.
`callback`	function	JavaScript method that's called upon completion of the method.

Tip: Try to give buttons short names. Scrolling is limited to the width of the button. If a button name is too long, scrolling can restart before the name finishes displaying.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testScrollCustomConsoleComponentButtonText();return false">
         Click here to scroll the button text on a custom console component</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testScrollCustomConsoleComponentButtonText() {
        //Scroll the custom console component button text from right to left
            sforce.console.scrollCustomConsoleComponentButtonText(500, 10, true,
function(result){
              if (result.success) {
               alert('The text scrolling starts!');
              } else {
               alert('Could not initiate the text scrolling!');
              }
             });
           }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if scrolling the button text was successful; `false` if scrolling the button text wasn't successful.

`selectMacro()`

Selects and displays a specific macro in the macro widget. This method is only available in API version 36.0 or later. This method isn't supported in Lightning Console.

Syntax

```
sforce.console.selectMacro(macroId:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method that is called when the method is completed
`macroID`	string	ID of the macro that's selected

Sample Code–Visualforce

```
<apex:page>
    <A HREF="#" onClick="openInWidget('0JZ00123456789A');return false">Click here to select
 a macro</A>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function openInWidget(id) {
            sforce.console.selectMacro(id);
        }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`cause`	string	Explanation of function failure, if applicable
`success`	boolean	`true` if selecting the macro was successful; `false` otherwise

`setCustomConsoleComponentButtonIconUrl()`

Sets the button icon URL of an application-level custom console component that's on a page. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.setCustomConsoleComponentButtonIconUrl(iconURL:String,
(optional)callback:Function)
```

Arguments

Name	Type	Description
`iconUrl`	string	A URL that points to an image that's used as a button to launch a custom console component.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testSetCustomConsoleComponentButtonIconUrl();return false">
         Click here to set the custom console component button icon</A>
```

```
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetCustomConsoleComponentButtonIconUrl() {

sforce.console.setCustomConsoleComponentButtonIconUrl('http://imageserver/img.png');
        }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if setting the button icon URL was successful; `false` if setting the button icon URL wasn't successful.

setCustomConsoleComponentButtonStyle()

Sets the style of a button used to launch an application-level custom console component that's on a page. This method is only available in API version 25.0 or later. This method isn't supported in Lightning Console.

Syntax

```
sforce.console.setCustomConsoleComponentButtonStyle(style:String, (optional)callback:
Function)
```

Arguments

Name	Type	Description
`style`	string	The style of a button used to launch a custom console component. The styles supported include font, font color, and background color. Font and font color isn't available for Internet Explorer® 7.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testSetCustomConsoleComponentButtonStyle();return false">
         Click here to set the style of a button used to launch a custom console
component</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
```

```
        function testSetCustomConsoleComponentButtonStyle() {
            sforce.console.setCustomConsoleComponentButtonStyle('background:red;');
        }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if setting the button style was successful; `false` if setting the button style wasn't successful.

setCustomConsoleComponentButtonText()

Sets the text on a button used to launch an application-level custom console component that's on a page. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.setCustomConsoleComponentButtonText(text:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`text`	string	Text that's displayed on a button used to launch a custom console component.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testSetCustomConsoleComponentButtonText();return false">
        Click here to set the text on a button used to launch a custom console component</A>


    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetCustomConsoleComponentButtonText() {
        //Change the custom console component button text to 'Hello World'
            sforce.console.setCustomConsoleComponentButtonText('Hello World');
        }
    </script>
</apex:page>
```

Response

Name	Type	Description
`success`	boolean	`true` if setting the button text was successful; `false` if setting the button text wasn't successful.

`setCustomConsoleComponentHeight()`

Sets the window height of an application-level custom console component that's on a page. This method is available in API version 32.0 or later.

Note: If this method is called from a popped out component in a Salesforce console where multi-monitor components is turned on, nothing will happen.

Syntax

```
sforce.console.setCustomConsoleComponentHeight( height:number, (optional)callback:Function)
```

Arguments

Name	Type	Description
`height`	number	The new height in pixels.
`callback`	function	Javascript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testSetCustomConsoleComponentHeight();return false">
         Click here to set the custom console component height to 100px</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetCustomConsoleComponentHeight() {
        // Set the custom console component height
            sforce.console.setCustomConsoleComponentHeight(100);
        }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the method call was successful; `false` otherwise.

`setCustomConsoleComponentVisible()`

Sets the window visibility of an application-level custom console component that's on a page. This method is available in API version 32.0 and later. In API version 31.0 and earlier, this method was called `setCustomConsoleComponentWindowVisible()`.

Syntax

```
sforce.console.setCustomConsoleComponentVisible(visible:Boolean,
(optional)callback:Function)
```

Arguments

Name	Type	Description
`visible`	boolean	`true` to make the custom console component window visible, `false` to hide the custom console component window.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testSetCustomConsoleComponentVisible();return false">
         Click here to make the custom console component window visible</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetCustomConsoleComponentVisible() {
        // Make the custom console component window visible
            sforce.console.setCustomConsoleComponentVisible(true);
        }
    </script>
</apex:page>
```

Response

Name	Type	Description
`success`	boolean	`true` if setting the button window visibility was successful; `false` if setting the button window visibility wasn't successful.

setCustomConsoleComponentWidth()

Sets the window width of an application-level custom console component that's on a page. This method is available in API version 32.0 or later.

Note: If this method is called from a popped out component in a Salesforce console where multi-monitor components is turned on, nothing will happen.

Syntax

```
sforce.console.setCustomConsoleComponentWidth( width:number, callback:Function)
```

Arguments

Name	Type	Description
width	number	The new width in pixels.
callback	function	Javascript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testSetCustomConsoleComponentWidth();return false">
         Click here to set the custom console component width to 100px</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetCustomConsoleComponentWidth() {
        // Set the custom console component width
            sforce.console.setCustomConsoleComponentWidth(100);
        }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
success	boolean	true if the method call was successful; false otherwise.

setCustomConsoleComponentPopoutable()

Sets a custom console component to be popped out or popped into a browser. To use this method, multi-monitor components must be turned on. This method is only available in API version 30.0 or later.

Syntax

```
sforce.console.setCustomConsoleComponentPopoutable(popoutable:Boolean,
(optional)callback:Function)
```

Arguments

Name	Type	Description
`popoutable`	boolean	If `true`, the component can be popped out or popped into a browser. If `false`, the component cannot be popped out or popped into a browser.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="enablePopout(); return false;">
         Click here to enable pop out or pop in functionality</A> <BR/>
    <A HREF="#" onClick="disablePopout(); return false;">
         Click here to disable pop out or pop in functionality</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function checkResult(result) {
          if (result.success) {
            alert('The method was successfully invoked.');
          } else {
            alert('Error while invoking this method. Check the browser developer console
for more information.');
          }
        }

        function enablePopout() {
          sforce.console.setCustomConsoleComponentPopoutable(true, checkResult);
        }

        function disablePopout() {
          sforce.console.setCustomConsoleComponentPopoutable(false, checkResult);
        }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if enabling pop out or pop in functionality for the component was successful; `false` otherwise.

`setCustomConsoleComponentWindowVisible()`

Sets the window visibility of an application-level custom console component that's on a page. This method is available in API versions 25.0 through 31.0. This method isn't supported in Lightning Console.

Note: If this method is called from a popped out component in a Salesforce console where multi-montior components is turned on, nothing will happen. Starting in API version 32.0, this method is no longer available and has been replaced by `setCustomConsoleComponentVisible()`. For more information, see `setCustomConsoleComponentVisible()`.

Syntax

```
sforce.console.setCustomConsoleComponentWindowVisible(visible:Boolean,
(optional)callback:Function)
```

Arguments

Name	Type	Description
`visible`	boolean	`true` to make the custom console component window visible, `false` to hide the custom console component window.
`callback`	function	JavaScript method that's called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>

    <A HREF="#" onClick="testSetCustomConsoleComponentWindowVisible();return false">
         Click here to make the custom console component window visible</A>

    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        function testSetCustomConsoleComponentWindowVisible() {
        //Make the custom console component window visible
            sforce.console.setCustomConsoleComponentWindowVisible(true);
        }
    </script>
</apex:page>
```

Response

Name	Type	Description
`success`	boolean	`true` if setting the button window visibility was successful; `false` if setting the button window visibility wasn't successful.

`setSidebarVisible()`

Shows or hides a console sidebar based on `tabId` and region. This method is available in API version 33.0 or later. This method isn't supported in Lightning Console.

Syntax

```
sforce.console.setSidebarVisible( visible:Boolean, (optional)tabId:String,
(optional)region:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`visible`	boolean	`true` to show the sidebar or `false` to hide the sidebar.
`tabId`	string	The ID of the tab on which to show or hide the sidebar.
`region`	string	The region on the console where the sidebar is located, such as left or right, top or bottom. Regions are represented as: • `sforce.console.Region.LEFT` • `sforce.console.Region.RIGHT` • `sforce.console.Region.TOP` • `sforce.console.Region.BOTTOM`
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
   <apex:includeScript value="/support/console/63.0/integration.js"/>
   <script type="text/javascript">

      var callback = function (result) {
         if (result.success) {
      alert('Congratulations!');
         }else {
      alert('something is wrong!');
         }
      };
         function setSidebarVisible() {
```

```
sforce.console.setSidebarVisible(true,'scc-st-1',sforce.console.Region.LEFT,callback);
      }

   </script>
    <A HREF="#" onClick="setSidebarVisible(); return false">SetSidebarToExpand</A>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if the method call was successful; `false` otherwise.

Methods for Push Notifications

Push notifications are visual indicators on lists and detail pages in a console that show when a record or field has changed during a user's session. For example, if two support agents are working on the same case, and one agent changes the `Priority`, a push notification appears to the other agent so he or she spots the change and doesn't duplicate the effort.

When administrators set up a Salesforce console, they choose when push notifications display, and which objects and fields trigger push notifications. Developers can use push notification methods to customize push notifications beyond the default visual indicators supplied by Salesforce. For example, developers can use the methods below to create personalized notifications about objects accessible to specific console users, thereby eliminating the need for email notifications.

Consider the following when using push notification methods:

- Push notification listener response is only available for the objects and fields selected to trigger push notifications for a console.
- When a Visualforce page includes a listener added by the `addPushNotificationListener()` method, the page receives notifications. The listener receives notifications when there is an update by any user to the objects selected for triggering console push notifications and the current user has access to the modified record. This functionality is slightly different from push notifications set up in the Salesforce user interface in that:
  - Listeners receive update notifications for changes made by all users.
  - When `Choose How Lists Refresh` is set to *`Refresh List Rows`* and the user is viewing an empty list view for an object set to trigger push notifications, a listener receives notifications for any record of that object created as well as any updates made to fields selected to trigger push notifications on the object.
  - When `Choose How Lists Refresh` is set to *`Refresh List`* and the user is viewing a list view for an object set to trigger push notifications, a listener receives notifications for any record of that object created and any updates made to fields selected to trigger push notifications, where the viewing user is the owner of the record.
  - The only way to stop receiving notifications is to remove listeners using the `removePushNotificationListener()` method.
- Push notifications aren't available in the console in Professional Edition.

IN THIS SECTION:

addPushNotificationListener()

Adds a listener for a push notification. A user can only register a listener once until he or she removes the listener, or the listener is removed by another user. This method is only available in API version 26.0 or later.

For more information on push notifications, see Methods for Push Notifications on page 272.

Syntax

```
sforce.console.addPushNotificationListener( objects: array, eventHandler:Function )
```

Arguments

Name	Type	Description
objects	array	Objects set to receive notifications.
eventHandler	function	JavaScript method called when there is a push notification.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var eventHandler = function (result) {
            alert('There is a push notification of object: ' + result.Id);
         };
         //Add a push notification listener for Case and Account
         sforce.console.addPushNotificationListener(['Case', 'Account'], eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method.

Name	Type	Description
id	string	The object ID of the push notification.

Name	Type	Description
`entityType`	string	The type of object included in the push notification. For example, Account or Contact. Objects available for push notifications are determined by the administrator who set up a Salesforce console.
`Type`	string	The field of the object included in the push notification. For example, the `Account Name` field on Account. Notifications occur when the field is either updated or created. Fields on objects available for push notifications are determined by the administrator who set up a Salesforce console.
`LastModifiedById`	string	The user ID of the user who last modified the object in the push notification.

`removePushNotificationListener()`

Removes a listener that gets added for a push notification. This method is only available in API version 26.0 or later.

For more information on push notifications, see Methods for Push Notifications on page 272.

Syntax

```
sforce.console.removePushNotificationListener((optional) callback:Function )
```

Arguments

Name	Type	Description
`callback`	function	A function called when the removal of the push notification listener completes.

Sample Code–Visualforce

```
<apex:page standardController="Case">

     <A HREF="#" onClick="testRemovePushNotification();return false">
         Click here to remove push notification</A>

<apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        function testRemovePushNotification() {
            sforce.console.removePushNotificationListener(removeSuccess);
         }
        var removeSuccess = function removeSuccess(result) {
            //Report whether removing the push notification listener is successful
            if (result.success == true) {
                alert('Removing push notification was successful');
            } else {
                alert('Removing push notification wasn't successful');
```

```
            }
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method.

Name	Type	Description
`success`	boolean	`true` if removing the push notification listener was successful; `false` if removing the push notification listener wasn't successful.

Methods for Console Events

JavaScript can be executed when certain types of events occur in a console, such as when a user closes a tab. The following standard events are supported:

Event	Description	Payload
`sforce.console.ConsoleEvent.OPEN_TAB`	Fired when a primary tab or subtab is opened. Available in API version 30.0 or later.	• `id` — The ID of the opened tab. • `objectId` — The object ID of the opened tab, if available.
`sforce.console.ConsoleEvent.CLOSE_TAB`	Fired when a primary tab or subtab with a specified ID in the `additionalParams` argument is closed. Or, fired when a primary tab or subtab with no specified ID is closed. Available in API version 30.0 or later.	• `id` — The ID of the closed tab. • `objectID` — The object ID of the closed tab, if available. Note: For some objects (such as Email and Case Comment), the tab is opened and closed immediately and no object ID is generated for the tab. In those cases, this field's value is equal to the parent's object ID. • `tabObjectId` — The object ID of the closed tab, if available. Note: `tabObjectId` is generally the same as `objectID`. However, for tabs that close upon submission, no `tabObjectId` is generated. In those cases, the value of this field is either empty or null. For an Email, the value is empty. For a Case Comment, the value is null.

Event	Description	Payload
sforce.console.ConsoleEvent.CONSOLE_LOGOUT	Delays the execution of logging out of a console when a user clicks **Logout**. When **Logout** is clicked: **1.** An overlay appears, which tells a user that logout is in progress. **2.** Callbacks are executed that have been registered by using sforce.console.ConsoleEvent.CONSOLE_LOGOUT **3.** Console logout logic is executed. If the callback contains synchronous blocking code, the console logout code isn't executed until the blocking code is executed. As a best practice, avoid synchronous blocking code or long code execution during logout. Available in API version 31.0 or later.	None

IN THIS SECTION:

addEventListener()

Adds a listener for a custom event type or a standard event type when the event is fired. This method adds a listener for custom event types in API version 25.0 or later; it adds a listener for standard event types in API version 30.0 or later.

For the list of standard events, see Methods for Console Events on page 275.

Syntax

```
sforce.console.addEventListener( eventType: String, eventListener:Function,
(optional)additionalParams:Object )
```

Arguments

Name	Type	Description
eventType	string	Custom event type for which eventListener listens.

Name	Type	Description
`eventListener`	function	JavaScript method called when an eventType is fired.
`additionalParams`	object	Optional parameters accepted by this method. The supported properties on this object are `tabId`: The ID of the tab to listen for the specified event. This argument is only available in API version 30.0 or later.

Sample Code API Version 25.0 or Later–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var listener = function (result) {
            alert('Message received from event: ' + result.message);
        };
        //Add a listener for the 'SampleEvent' event type
        sforce.console.addEventListener('SampleEvent', listener);
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`message`	string	The message which is sent with the fired event. If the response is from a custom keyboard shortcut, the `message` includes the following information on which the browser is focused, in this order: **1.** Object ID of the primary tab **2.** ID of the primary tab **3.** Object ID of the subtab **4.** ID of the subtab

Sample Code API Version 30.0 or Later–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        var onEnclosingPrimaryTabClose = function (result) {
            alert('The enclosing primary tab is about to be closed. Tab ID: ' + result.id
 + ', Object ID: ' + (result.objectId ? result.objectId : 'not available'));
        };
```

```
        //Add a listener to handle the closing of the enclosing primary tab
        sforce.console.getEnclosingPrimaryTabId(function (result) {
            if (result.id) {
                sforce.console.addEventListener(sforce.console.ConsoleEvent.CLOSE_TAB,
                onEnclosingPrimaryTabClose, { tabId : result.id });
            } else {
                alert('Could not find an enclosing primary TAB!');
            }
        });
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`message`	string	The message which is sent with the fired event. If the response is from a console event, the `message` includes payload details as described in Methods for Console Events on page 275. If the response is from a custom keyboard shortcut, the `message` includes the following information on which the browser is focused, in this order: 1. Object ID of the primary tab 2. ID of the primary tab 3. Object ID of the subtab 4. ID of the subtab

`fireEvent()`

Fires a custom event. This method is only available in API version 25.0 or later.

Syntax

```
sforce.console.fireEvent( eventType:String, message:String, (optional)callback:Function
)
```

Arguments

Name	Type	Description
`eventType`	string	The type of custom event to fire.
`message`	string	The message which is sent with the fired event.

Name	Type	Description
`callback`	function	JavaScript method called when the custom event is fired.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">

        <A HREF="#" onClick="testFireEvent(); return false;">
            Click here to fire an event of type 'SampleEvent'</A>

        var callback = function(result) {
            if (result.success) {
                   alert('The custom event is fired!');
            } else {
                   alert('The custom event could not be fired!');
            }
         };

        function testFireEvent() {
         //Fire an event of type 'SampleEvent'
         sforce.console.fireEvent('SampleEvent', 'EventMessage', callback);
         }
    </script>
</apex:page>
```

Response

This method is asynchronous, so it returns its response in an object in a callback method. The response object contains the following field:

Name	Type	Description
`success`	boolean	`true` if firing the event is successful, `false` if firing the event wasn't successful.

`removeEventListener()`

Removes a listener for a custom event type or a standard event type. This method removes a listener for custom event types in API version 25.0 or later; it removes a listener for standard event types in API version 30.0 or later.

For the list of standard events, see Methods for Console Events on page 275.

Syntax

```
sforce.console.removeEventListener( eventType: String, eventListener:Function,
(optional)additionalParams:Object )
```

Arguments

Name	Type	Description
`eventType`	string	Event type for which eventListener is removed.
`eventListener`	function	Event listener to remove.
`additionalParams`	object	Optional parameters accepted by this method. The supported properties on this object are `tabId`: The ID of the tab to remove the listener for the specified event. This argument is only available in API version 30.0 or later.

Sample Code API Version 25.0 or Later–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
         <A HREF="#" onClick="testRemoveEventListener(); return false;">
              Click here to remove an event listener for the 'SampleEvent' event type</A>


    <script type="text/javascript">
        var listener = function (result) {
            alert('Message received from event: ' + result.message);
        };
        //Add a listener for the 'SampleEvent' event type
        sforce.console.addEventListener('SampleEvent', listener);

        function testRemoveEventListener() {
            sforce.console.removeEventListener('SampleEvent', listener);
        }
    </script>
</apex:page>
```

Response

None

Sample Code API Version 30.0 or Later–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
         <A HREF="#" onClick="testRemoveEventListener(); return false;">
              Click here to remove an event listener for the console 'CLOSE_TAB' event
type</A>

    <script type="text/javascript">
        var tabId;

        var onEnclosingPrimaryTabClose = function (result) {
            alert('The enclosing primary tab is about to be closed. Tab ID: ' + result.id
 + ',
```

```
                    Object ID: ' + (result.objectId ? result.objectId : 'not available'));

        };

        //Add a listener to handle the closing of the enclosing primary tab
        sforce.console.getEnclosingPrimaryTabId(function (result) {
            if (result.id) {
                tabId = result.id;
                sforce.console.addEventListener(sforce.console.ConsoleEvent.CLOSE_TAB,
    onEnclosingPrimaryTabClose, { tabId : tabId });
            } else {
                alert('Could not find an enclosing primary TAB!');
            }
        });

        function testRemoveEventListener() {
            sforce.console.removeEventListener(sforce.console.ConsoleEvent.CLOSE_TAB,
                        onEnclosingPrimaryTabClose, { tabId : tabId });
        }
    </script>
</apex:page>
```

Response

None

Methods for Chat

Connect with customers or website visitors in real time through Web-based chat.

Note: These methods in Salesforce Classic don't work for chats routed with Omni-Channel. Chats with Omni-Channel routing use the Methods for Omni-Channel. If you're using Lightning Experience, use the Methods for Omni-Channel in Lightning Experience.

Important: The legacy chat product is in maintenance-only mode, and we won't continue to build new features. You can continue to use it, but we no longer recommend that you implement new chat channels. Instead, you can modernize your customer communication with Messaging for In-App and Web. Messaging offers many of the chat features that you love plus asynchronous conversations that can be picked back up at any time.

IN THIS SECTION:

getAgentInput()

Returns the string of text which is currently in the agent's text input area in the chat log of a chat with a specific chat key. Available in API version 29.0 or later.

getAgentState()

Returns the agent's current Chat status, such as Online, Away, or Offline. Available in API version 29.0 or later.

getChatLog()

Returns the chat log of a chat associated with a specific chat key. Available in API version 29.0 or later.

getChatRequests()

Returns the chat keys of the chat requests that have been assigned to an agent. Available in API version 29.0 or later.

getDetailsByChatKey()

Returns the details of the chat associated with a specific chat key. Available in API version 29.0 or later.

getDetailsByPrimaryTabId()

Returns the details of the chat associated with a specific primary tab ID. Available in API version 29.0 or later.

getEngagedChats()

Returns the chat keys of the chats in which the agent is currently engaged. Available in API version 29.0 or later.

getMaxCapacity()

Returns the maximum chat capacity for the current agent, as specified in the agent's assigned agent configuration. Available in API version 29.0 or later.

initFileTransfer()

Initiates the process of transferring a file from a customer to an agent. Available in API version 31.0 or later.

onAgentSend()

Registers a function to call when an agent sends a chat message through the Salesforce console. This method intercepts the message and occurs before it is sent to the chat visitor. Available in API version 29.0 or later.

onAgentStateChanged()

Registers a function to call when agents change their Chat status, such as from Online to Away. Available in API version 29.0 or later.

onChatCanceled()

Registers a function to call when a chat visitor cancels a chat request. Available in API version 29.0 or later.

onChatCriticalWaitState()

Registers a function to call when a chat becomes critical to answer or a waiting chat is answered. Available in API version 29.0 or later.

onChatDeclined()

Registers a function to call when an agent declines a chat request. Available in API version 29.0 or later.

onChatEnded()

Registers a function to call when an engaged chat ends. Available in API version 29.0 or later.

onChatRequested()

Registers a function to call when an agent receives a chat request. Available in API version 29.0 or later.

onChatStarted()

Registers a function to call when an agent starts a new chat with a customer. Available in API version 29.0 or later.

onChatTransferredOut()

Registers a function to call when an engaged chat is transferred out to another agent. Available in API version 29.0 or later.

onCurrentCapacityChanged()

Registers a function to call when an agent's capacity for accepting chats changes—for example, if an agent accepts a new chat, ends a currently engaged chat, or otherwise changes the number of chats to which they are assigned, or if a chat request is pushed to their chat queue. Available in API version 29.0 or later.

onCustomEvent()

Registers a function to call when a custom event takes place during a chat. Available in API version 29.0 or later.

onFileTransferCompleted()

Registers a function to call when a file is transferred from a customer to an agent. Available in API version 31.0 or later.

onNewMessage()

Registers a function to call when a new message is sent from a customer, agent, or supervisor. Available in API version 29.0 or later.

onTypingUpdate()

Registers a function to call when the customer's text in the chat window changes. If Sneak Peek is enabled, this function is called whenever the customer edits the text in the chat window. If Sneak Peek is not enabled, this function is called whenever a customer starts or stops typing in the chat window. Available in API version 29.0 or later.

sendCustomEvent()

Sends a custom event to the client-side chat window for a chat with a specific chat key. Available in API version 29.0 or later.

sendMessage()

Sends a new chat message from the agent to a chat with a specific chat key. Available in API version 29.0 or later.

setAgentInput()

Sets the string of text in the agent's text input area in the chat log of a chat with a specific chat key.Available in API version 29.0 or later.

setAgentState()

Sets an agent's Chat status, such as Online, Away, or Offline. Available in API version 29.0 or later.

Methods for Chat Visitors

There are a few methods available that you can use to customize the visitor experience for Chat in a custom Visualforce chat window. These methods apply to Salesforce Classic only.

`acceptChat()`

Accepts a chat request. Available in API version 29.0 or later. This method isn't supported with Omni-Channel in API version 37.0 or later.

Syntax

```
sforce.console.chat.acceptChat(chatKey:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The chat key for the chat request you wish to accept.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testAcceptChat();return false;">Accept Chat</a>

    <script type="text/javascript">
        function testAcceptChat() {
           //Get the value for 'myChatKey'from the getChatRequests() or onChatRequested()
 methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            sforce.console.chat.acceptChat(chatKey, acceptSuccess);
        }

        function acceptSuccess(result) {
            //Report whether accepting the chat was succesful
            if (result.success == true) {
                alert('Accepting the chat was successful');
            } else {
                alert('Accepting the chat was not successful');
            }
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if accepting the chat was successful; `false` if accepting the chat wasn't successful.

`cancelFileTransferByAgent()`

Indicates that a file transfer request has been canceled by an agent. Available in API version 31.0 or later.

Syntax

```
sforce.console.chat.cancelFileTransferByAgent(chatKey:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The chat key for the chat for which the agent canceled the file transfer request.
`callback`	function	JavaScript method that is called when the method is completed.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testCancelFileTransfer();return false;">Cancel file transfer</a>


    <script type="text/javascript">
        function testCancelFileTransfer() {
           //Gets the value for 'myChatKey'from the getChatRequests() or onChatRequested()

              methods.
            //These values are for example purposes only.
            var chatKey = 'myChatKey';
            sforce.console.chat.cancelFileTransferByAgent(chatKey, fileSuccess);
        }

        function fileSuccess(result) {
            //Report whether canceling was successful
            if (result.success == true) {
                alert('Canceling file transfer was successful.');
            } else {
                alert('Canceling file transfer was not successful.');
            }
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if canceling the file transfer request was successful; `false` if canceling the file transfer request wasn't successful.

`declineChat()`

Declines a chat request. Available in API version 29.0 or later. This method isn't supported with Omni-Channel in API version 37.0 or later.

Syntax

```
sforce.console.chat.declineChat(chatKey:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The chat key for the request you wish to decline.

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
                <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testDeclineChat();return false;">Decline Chat</a>

    <script type="text/javascript">
        function testDeclineChat() {
           //Get the value for 'myChatKey'from the getChatRequests() or onChatRequested()
 methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            sforce.console.chat.declineChat(chatKey, declineSuccess);
        }


        function declineSuccess(result) {
            //Report whether declining the chat was succesful
            if (result.success == true) {
                alert('Declining the chat was successful');
            } else {
                alert('Declining the chat was not successful');
            }
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if declining the event was successful; `false` if declining the event wasn't successful.

`endChat()`

Ends a chat in which an agent is currently engaged. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.endChat(chatKey:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The chat key for the engaged chat you wish to end.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testEndChat();return false;">End Chat</a>

    <script type="text/javascript">
        function testEndChat() {
            //Get the value for 'myChatKey'from the getEngagedChats() or onChatStarted()
methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            sforce.console.chat.endChat(chatKey, endSuccess);
        }

        function endSuccess(result) {
            //Report whether ending the chat was succesful
            if (result.success == true) {
                alert('Ending the chat was successful');
            } else {
                alert('Ending the chat was not successful');
            }
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if ending the chat was successful; `false` if ending the chat wasn't successful.

`getAgentInput()`

Returns the string of text which is currently in the agent's text input area in the chat log of a chat with a specific chat key. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getAgentInput(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` associated with the chat for which to retrieve the agent's input text.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetAgentInput();">Get Agent Input</a>

    <script type="text/javascript">

        function testGetAgentInput() {
            //Get the value for 'myChatKey'from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            sforce.console.chat.getAgentInput(chatKey, getAgentInputSuccess);
        }

        function getAgentInputSuccess(result) {
            //Report whether getting the agent's input was successful
            if (result.success == true) {
                agentInput = result.text;
                alert('The text in the agent input is: ' + agentInput);
            } else {
                alert('Getting the agent input was not successful');
            }
        };


    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`text`	String	The text that is currently in an agent's text input area.

Name	Type	Description
`success`	Boolean	`true` if getting the agent's input was successful; `false` if getting the agent's input wasn't successful.

`getAgentState()`

Returns the agent's current Chat status, such as Online, Away, or Offline. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getAgentState(callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetAgentState();return false;">Get Agent State</a>

    <script type="text/javascript">
        function testGetAgentState() {
            sforce.console.chat.getAgentState(function(result) {
                if (result.success) {
                    alert('Agent State:' + result.state);
                } else {
                    alert('getAgentState has failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`state`	String	String representing the current agent state—for example, Online, Away, or Offline.
`success`	Boolean	`true` if getting the agent's Chat status was successful; `false` if getting the agent's Chat status wasn't successful.

getChatLog()

Returns the chat log of a chat associated with a specific chat key. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getChatLog(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
chatKey	String	The chatKey associated with the chat for which to retrieve the chat log.
callback	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetChatLog();">Get Chat Log</a>

    <script type="text/javascript">

        function testGetChatLog() {
            //Get the value for 'myChatKey'from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            sforce.console.chat.getChatLog(chatKey, getChatLogSuccess);
        }

        function getChatLogSuccess(result) {
            //Report whether getting the chat log was succesful
            if (result.success == true) {
                chatLogMessage = result.messages[0].content;
                alert('The first message in this chatLog is: ' + chatLogMessage);
            } else {
                alert('Getting the chat log was not successful');
            }
        };


    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following fields:

Name	Type	Description
`customEvents`	Array of `customEvent` objects	An array of custom event objects representing the custom events that occurred during a chat.
`messages`	Array of `message` objects	An array of chat message objects containing all of the chat messages from the chat log.
`success`	Boolean	`true` if getting the chat log was successful; `false` if getting the chat log wasn't successful.

`customEvent`

The `customEvent` object contains a single event from the chat log and the following properties:

Property	Type	Description
`source`	String	The person who initiated the custom event, either the chat visitor or the agent.
`type`	String	The type of custom event that occurred.
`data`	String	The data of the custom event that was sent to the chat; corresponds to the `data` argument of the `liveagent.chasitor.sendCustomEvent()` method used to send this event from the chat window.
`timestamp`	Date/Time	The date and time a custom event was received.

`message`

The `message` object contains a single chat message from the chat log and the following properties:

Property	Type	Description
`content`	String	The text content of a message in the chat log.
`name`	String	The name of the user who sent the message in the chat log. This appears exactly as it is displayed in the chat log.
`type`	String	The type of message that was received, such as Agent or Visitor.
`timestamp`	Date/Time	The date and time the chat message was received.

`getChatRequests()`

Returns the chat keys of the chat requests that have been assigned to an agent. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getChatRequests(callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetChatRequests();return false;">Get Chat Requests</a>

    <script type="text/javascript">
        function testGetChatRequests() {
            sforce.console.chat.getChatRequests(function(result) {
                if (result.success) {
                    alert('Number of Chat Requests ' + result.chatKey.length);
                } else {
                    alert('getChatRequests has failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	Array	Array of `chatKey` values, one for each of the current chat requests.
`success`	Boolean	`true` if getting chat requests was successful; `false` if getting chat requests wasn't successful.

`getDetailsByChatKey()`

Returns the details of the chat associated with a specific chat key. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getDetailsByChatKey(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` associated with the chat for which to retrieve details.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetDetailsByChatKey();">Get Chat Details</a>

    <script type="text/javascript">

        function testGetDetailsByChatKey() {
            //Get the value for 'myChatKey' from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            sforce.console.chat.getDetailsByChatKey(chatKey, getDetailsSuccess);
        }

        function getDetailsSuccess(result) {
            //Report whether accepting the chat was succesful
            if (result.success == true) {
                ipAddress = result.details.ipAddress;
                alert('The Visitor IP Address for this chat is: ' + ipAddress);
            } else {
                alert('Getting the details was not successful');
            }
        };


    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`primaryTabId`	String	The ID of the primary tab associated with the chat.
`details`	Object	An object that contains all the details for a chat associated with a particular primary tab.
`success`	Boolean	`true` if retrieving the details was successful; `false` if retrieving the details wasn't successful.

details

The `details` object contains the following properties:

Property	Type	Description
`acceptTime`	Date/Time	The date and time an agent accepted the chat request.
`breadcrumbs`	Array of `breadcrumb` objects	An array of `breadcrumb` objects representing a list of Web pages visited by the visitor before and during the chat.
`chatKey`	String	The chat key associated with the chat.
`customDetails`	Array of `customDetail` objects	An array of `customDetail` objects that represent custom details that have been passed in to this chat via the Deployment API or Pre-Chat Form API.
`geoLocation`	Object	An object representing the details of a chat visitor's location, derived from a geoIP lookup on the chat visitor's IP address.
`ipAddress`	String	The IP address of the chat visitor in dot-decimal format.
`isEnded`	Boolean	Specifies whether a chat has ended (`true`) or not (`false`).
`isEngaged`	Boolean	Specifies whether a chat is currently engaged (`true`) or not (`false`).
`isPushRequest`	Boolean	Specifies whether a chat was routed to an agent through a push-based routing method such as Least Active or Most Available (`true`) or not (`false`).
`isTransferringOut`	Boolean	Specifies whether a chat is currently in the process of being transferred to another agent (`true`) or not (`false`).
`liveChatButtonId`	String	The 15-digit record ID for the chat button from which the chat request originated.
`liveChatDeploymentId`	String	The 15-digit record ID for the deployment from which the chat request originated.
`name`	String	The name of the chat visitor.
`requestTime`	Date/Time	The date and time the chat was requested.
`visitorInfo`	Object	An object containing information about the visitor's web browser.

breadcrumb

A breadcrumb represents a Web page viewed by a chat visitor. The `breadcrumb` object contains the following properties:

Property	Type	Description
`location`	String	The URL of a Web page viewed by a chat visitor.
`time`	Date/Time	The date and time a chat visitor visited a specific breadcrumb URL.

customDetail

Custom details are details have been passed into the chat through the Deployment API or Pre-Chat Form API. The `customDetail` object contains the following properties:

Property	Type	Description
label	String	The name of the custom detail as specified in the Deployment API or Pre-Chat Form API.
value	String	The value of the custom detail as specified in the Deployment API or Pre-Chat Form API.
transcriptFields	Array of Strings	The names of fields where the customer's details on the chat transcript are saved.
entityMaps	Array of entityMap objects	An array of pre-created records used for mapping custom detail information.

entityMap

Entities are records that are created when a customer starts a chat with an agent. You can use the API to auto-populate these records with customer details. The `entityMap` object contains the following properties:

Property	Type	Description
entityName	String	The record to search for or create.
fieldName	String	The name of the field associated with the details.
isFastFillable	Boolean	Specifies whether the value can be used to populate the field when an agent creates or edits a record (`true`) or not (`false`) (Live Agent console only).
isAutoQueryable	Boolean	If you're using the Live Agent console, specifies whether to perform a a SOSL query (in the Live Agent console) (`true`) or not (`false`) to find records with a `fieldName` containing the value. If you're using the Salesforce console, specifies whether to perform a SOQL query (in the Salesforce console) (`true`) or not (`false`) to find records with a `fieldName` containing the value.
isExactMatchable	Boolean	Specifies whether to only search for records that have fields exactly matching the field `fieldName` (`true`) or not (`false`).

geoLocation

The `geoLocation` object represents the details of a chat visitor's location. It contains the following properties:

Property	Type	Description
city	String	The name of the chat visitor's city.
countryCode	String	The two-digit ISO-3166 country code for the chat visitor's country.

Property	Type	Description
`countryName`	String	The name of chat visitor's country.
`latitude`	String	The chat visitor's approximate latitude.
`longitude`	String	The chat visitor's approximate longitude.
`organization`	String	The organization name of the chat visitor's internet service provider.
`region`	String	The chat visitor's region, such as state or province.

`visitorInfo`

The `visitorInfo` object represents information about the visitor's web browser. It contains the following properties:

Property	Type	Description
`browserName`	String	The name and version of the chat visitor's web browser.
`language`	String	The language of the chat visitor's web browser.
`originalReferrer`	String	The original URL of the Web page from which the chat visitor requested a chat.
`screenResolution`	String	The screen resolution of the chat visitor's computer, as passed by the chat visitor's browser.
`sessionKey`	String	the sessionKey of the visitor which will ultimately be stored on the LiveChatVisitor record as a unique reference to this live chat visitor

`getDetailsByPrimaryTabId()`

Returns the details of the chat associated with a specific primary tab ID. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getDetailsByPrimaryTabId(primaryTabId:String, callback:Function)
```

Arguments

Name	Type	Description
`primaryTabId`	String	The ID of the primary tab associated with the chat for which to retrieve details.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetDetailsByPrimaryTabId();">Get Chat Details</a>
```

```
    <script type="text/javascript">

        function testGetDetailsByPrimaryTabId() {
            //Get the value for 'myPrimaryTabId'from the getPrimaryTabIds() or
getEnclosingPrimaryTabId() methods.
            //These values are for example purposes only
            var primaryTabId = 'myPrimaryTabId';
           sforce.console.chat.getDetailsByPrimaryTabId(primaryTabId, getDetailsSuccess);

        }

        function getDetailsSuccess(result) {
            //Report whether accepting the chat was succesful
            if (result.success == true) {
                console.log(result);
                chatKey = result.details.chatKey;
                alert('The chatKey for this chat is: ' + chatKey);
            } else {
                alert('Getting the details was not Succesful');
            }
        };

    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`primaryTabId`	String	The ID of the primary tab associated with the chat.
`details`	Object	An object that contains all the details for a chat associated with a particular primary tab.
`success`	Boolean	`true` if retrieving the details was successful; `false` if retrieving the details wasn't successful.

`details`

The `details` object contains the following properties:

Property	Type	Description
`acceptTime`	Date/Time	The date and time an agent accepted the chat request.
`breadcrumbs`	Array of `breadcrumb` objects	An array of `breadcrumb` objects representing a list of Web pages visited by the visitor before and during the chat.

Property	Type	Description
`chatKey`	String	The chat key associated with the chat.
`customDetails`	Array of `customDetail` objects	An array of `customDetail` objects that represent custom details that have been passed in to this chat via the Deployment API or Pre-Chat Form API.
`geoLocation`	Object	An object representing the details of a chat visitor's location, derived from a geoIP lookup on the chat visitor's IP address.
`ipAddress`	String	The IP address of the chat visitor in dot-decimal format.
`isEnded`	Boolean	Specifies whether a chat has ended (`true`) or not (`false`).
`isEngaged`	Boolean	Specifies whether a chat is currently engaged (`true`) or not (`false`).
`isPushRequest`	Boolean	Specifies whether a chat was routed to an agent through a push-based routing method such as Least Active or Most Available (`true`) or not (`false`).
`isTransferringOut`	Boolean	Specifies whether a chat is currently in the process of being transferred to another agent (`true`) or not (`false`).
`liveChatButtonId`	String	The 15-digit record ID for the chat button from which the chat request originated.
`liveChatDeploymentId`	String	The 15-digit record ID for the deployment from which the chat request originated.
`name`	String	The name of the chat visitor.
`requestTime`	Date/Time	The date and time the chat was requested.
`visitorInfo`	Object	An object containing information about the visitor's web browser.

`breadcrumb`

A breadcrumb represents a Web page viewed by a chat visitor. The `breadcrumb` object contains the following properties:

Property	Type	Description
`location`	String	The URL of a Web page viewed by a chat visitor.
`time`	Date/Time	The date and time a chat visitor visited a specific breadcrumb URL.

`customDetail`

Custom details are details that have been passed into the chat through the Deployment API or Pre-Chat Form API. The `customDetail` object contains the following properties:

Property	Type	Description
`label`	String	The name of the custom detail as specified in the Deployment API or Pre-Chat Form API.
`value`	String	The value of the custom detail as specified in the Deployment API or Pre-Chat Form API.

Property	Type	Description
`transcriptFields`	Array of Strings	The names of fields where the customer's details on the chat transcript are saved.
`entityMaps`	Array of `entityMap` objects	An array of pre-created records used for mapping the custom detail information.

`entityMap`

Entities are records that are created when a customer starts a chat with an agent. You can use the API to auto-populate these records with customer details. The `entityMap` object contains the following properties:

Property	Type	Description
`entityName`	String	The record to search for or create.
`fieldName`	String	The name of the field associated the details.
`isFastFillable`	Boolean	Specifies whether the value can be used to populate the field when an agent creates or edits a record (`true`) or not (`false`) (Live Agent console only).
`isAutoQueryable`	Boolean	If you're using the Live Agent console, specifies whether to perform a a SOSL query (in the Live Agent console) (`true`) or not (`false`) to find records with a `fieldName` containing the value. If you're using the Salesforce console, specifies whether to perform a SOQL query (in the Salesforce console) (`true`) or not (`false`) to find records with a `fieldName` containing the value.
`isExactMatchable`	Boolean	Specifies whether to only search for records that have fields exactly matching the field `fieldName` (`true`) or not (`false`).

`geoLocation`

The `geoLocation` object represents the details of a chat visitor's location. It contains the following properties:

Property	Type	Description
`city`	String	The name of the chat visitor's city.
`countryCode`	String	The two-digit ISO-3166 country code for the chat visitor's country.
`countryName`	String	The name of chat visitor's country.
`latitude`	String	The chat visitor's approximate latitude.
`longitude`	String	The chat visitor's approximate longitude.
`organization`	String	The organization name of the chat visitor's internet service provider.
`region`	String	The chat visitor's region, such as state or province.

`visitorInfo`

The `visitorInfo` object represents information about the visitor's web browser. It contains the following properties:

Property	Type	Description
`browserName`	String	The name and version of the chat visitor's web browser.
`language`	String	The language of the chat visitor's web browser.
`originalReferrer`	String	The original URL of the Web page from which the chat visitor requested a chat.
`screenResolution`	String	The screen resolution of the chat visitor's computer, as passed by the chat visitor's browser.
`sessionKey`	String	the sessionKey of the visitor which will ultimately be stored on the LiveChatVisitor record as a unique reference to this live chat visitor

`getEngagedChats()`

Returns the chat keys of the chats in which the agent is currently engaged. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getEngagedChats(callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetEngagedChats();return false;">Get Engaged Chats</a>

    <script type="text/javascript">
        function testGetEngagedChats() {
            sforce.console.chat.getEngagedChats(function(result) {
                if (result.success) {
                    alert('Number Engaged Chats: ' + result.chatKey.length);
                } else {
                    alert('getEngagedChats has failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	array	Array of `chatKey` values, one for each of the currently engaged chats.
`success`	boolean	`true` if getting engaged chats was successful; `false` if getting engaged chats wasn't successful.

`getMaxCapacity()`

Returns the maximum chat capacity for the current agent, as specified in the agent's assigned agent configuration. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.getMaxCapacity(callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetMaxCapacity();return false;">Get Max Capacity</a>

    <script type="text/javascript">
        function testGetMaxCapacity() {
            sforce.console.chat.getMaxCapacity(function(result) {
                if (result.success) {
                    alert('max capacity '+result.count);
                } else {
                    alert('getMaxCapacity failed, agent my not be online');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`count`	integer	Agent's current, maximum chat capacity.
`success`	boolean	`true` if getting the agent's capacity was successful; `false` if getting the agent's capacity wasn't successful.

`initFileTransfer()`

Initiates the process of transferring a file from a customer to an agent. Available in API version 31.0 or later.

Syntax

```
sforce.console.chat.initFileTransfer(chatKey:String, entityId:String,
(optional)callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The chat key for the chat the file is transferred from.
`entityId`	String	The ID of the transcript object to attach the transferred file to.
`callback`	function	JavaScript method that is called when the method is completed.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testInitFileTransfer();return false;">Init file transfer</a>

    <script type="text/javascript">
        function testInitFileTransfer() {
           //Gets the value for 'myChatKey'from the getChatRequests() or onChatRequested()
 methods.
            //These values are for example purposes only.
            var chatKey = 'myChatKey';  var entityId = 'myEntityId';
            sforce.console.chat.initFileTransfer(chatKey, entityId, fileSuccess);
        }

        function fileSuccess(result) {
            //Reports whether initiating the file transfer was successful.
            if (result.success == true) {
                alert('Initiating file transfer was successful.');
            } else {
                alert('Initiating file transfer was not successful.');
            }
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if the request to transfer a file was sent successfully; `false` if the request wasn't sent successfully. Note: A value of `true` doesn't necessarily mean that the file was successfully transferred to an agent. Rather, it indicates that the request to begin a file transfer was sent successfully.

`onAgentSend()`

Registers a function to call when an agent sends a chat message through the Salesforce console. This method intercepts the message and occurs before it is sent to the chat visitor. Available in API version 29.0 or later.

Note: This method is only called when an agent sends a message through the chat window interface. This method doesn't apply when a `sendMessage()` method is called in the API.

Syntax

```
sforce.console.chat.onAgentSend(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` associated with the chat for which to call a function when the agent sends a message.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            var theMessage = result.content;
            alert('The agent is attempting to send the following message: ' +
result.content);
            sforce.console.chat.sendMessage(chatKey, theMessage)
            alert('The following message has been sent: ' + theMessage);
        }
        //Get the value for 'myChatKey' from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
        //These values are for example purposes only
```

```
        var chatKey = 'myChatKey';
        sforce.console.chat.onAgentSend(chatKey, eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`content`	String	The text of the agent's message.
`name`	String	The name of the agent who is attempting to send the message as it appears in the chat log.
`type`	String	The type of message that was received—for example, agent.
`timestamp`	Date/Time	The date and time the agent attempted to send the chat message.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onAgentStateChanged()`

Registers a function to call when agents change their Chat status, such as from Online to Away. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onAgentStateChanged(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when the agent's Chat status has changed.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert("Agent's State has Changed to: " + result.state);
        };
        sforce.console.chat.onAgentStateChanged(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`state`	String	String that represents the agent's current Chat status—for example, Online, Away, or Offline. When an agent switches from Offline to Away, you may see two returned values (Online then Away) instead of one (Away).
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onChatCanceled()`

Registers a function to call when a chat visitor cancels a chat request. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onChatCanceled(callback:Function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
<apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
           alert('The chat request has been canceled for this chatKey: ' + result.chatKey);

        }
        sforce.console.chat.onChatCanceled(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	string	The chat key for the chat request that has been canceled.

onChatCriticalWaitState()

Registers a function to call when a chat becomes critical to answer or a waiting chat is answered. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onChatCanceled(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
chatKey	String	The chatKey associated with the chat for which the critical wait state has changed.
callback	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('This chat has reached a critical wait');
        }
        //Get the value for 'myChatKey' from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
        //These values are for example purposes only
        var chatKey = 'myChatKey';
        sforce.console.chat.onChatCriticalWaitState(chatKey, eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
state	Boolean	Indicates whether the chat is in critical wait state (true) or not (false).

onChatDeclined()

Registers a function to call when an agent declines a chat request. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onChatDeclined(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when a chat request is declined.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
           alert('A chat request with this chatKey has been declined: ' + result.chatKey);

        }
        sforce.console.chat.onChatDeclined(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	String	The chat key for the chat request that has been declined.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onChatEnded()`

Registers a function to call when an engaged chat ends. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onChatEnded(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when an engaged chat ends.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
```

```
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('A chat with this chatKey has ended: ' + result.chatKey);
        }
        sforce.console.chat.onChatEnded(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	String	The chat key for the engaged chat that has ended.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onChatRequested()`

Registers a function to call when an agent receives a chat request. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onChatRequested(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when a chat request is assigned to an agent.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('There is a new incoming chat request with this chatKey: ' +
result.chatKey);
        }
        sforce.console.chat.onChatRequested(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	String	The chat key for the incoming chat request.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onChatStarted()`

Registers a function to call when an agent starts a new chat with a customer. Available in API version 29.0 or later.

Usage

Syntax

```
sforce.console.chat.onChatStarted(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when a chat request is accepted and becomes an engaged chat.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('A new engaged chat has started for this chatKey: ' + result.chatKey);
        }
        sforce.console.chat.onChatStarted(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	String	The chat key for the chat request that has become an engaged chat.

Name	Type	Description
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onChatTransferredOut()`

Registers a function to call when an engaged chat is transferred out to another agent. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onChatTransferredOut(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when a chat has been successfully transferred out to another agent.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('A chat with this chatKey has been transferred out: ' + result.chatKey);

        }
        sforce.console.chat.onChatTransferredOut(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`chatKey`	String	The chat key for the chat that has been transferred.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onCurrentCapacityChanged()`

Registers a function to call when an agent's capacity for accepting chats changes—for example, if an agent accepts a new chat, ends a currently engaged chat, or otherwise changes the number of chats to which they are assigned, or if a chat request is pushed to their chat queue. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onCurrentCapacityChanged(eventHandler:Function)
```

Arguments

Name	Type	Description
`eventHandler`	function	JavaScript method called when the agent's capacity for accepting chats has changed.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('Capacity Changed. Current Requests + Engaged Chats is now: ' +
result.count);
        }
        sforce.console.chat.onCurrentCapacityChanged(eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`count`	integer	The number of chats in which the agent is currently engaged plus the number of chat requests currently assigned to the agent.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onCustomEvent()`

Registers a function to call when a custom event takes place during a chat. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onCustomEvent(chatKey:String, type:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` associated with the chat for which to call a function when a custom event takes place.
`type`	String	The name of the custom event you want to listen for. This should match the name of the custom event sent from the chat window.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('A new custom event has been received of type ' + result.type + ' and
with data: ' + result.data );
        }
        //Get the value for 'myChatKey' from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
        //These values are for example purposes only
        var chatKey = 'myChatKey';
        var type = 'myCustomEventType';
        sforce.console.chat.onCustomEvent(chatKey, type, eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`type`	String	The type of the custom event that was sent to this chat; corresponds to the `type` argument of the `liveagent.chasitor.sendCustomEvent()` method used to send this event from the chat window.
`data`	String	The data of the custom event that was sent to this chat; corresponds to the `data` argument of the `liveagent.chasitor.sendCustomEvent()` method used to send this event from the chat window.
`source`	String	The source of the custom event that was sent to this chat; corresponds to either the agent or the chat visitor, depending on who triggered the custom event.
`timestamp`	Date/Time	The time and date the event was received.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

onFileTransferCompleted()

Registers a function to call when a file is transferred from a customer to an agent. Available in API version 31.0 or later.

Syntax

```
sforce.console.chat.onFileTransferCompleted(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
chatKey	String	The chat key for the chat the file was transferred from.
callback	function	JavaScript method that is called when the method is complete.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testOnFileComplete();return false;">test on file transfer
complete</a>

    <script type="text/javascript">
        function testOnFileComplete() {
           //Gets the value for 'myChatKey'from the getChatRequests() or onChatRequested()
 methods.
            //These values are for example purposes only.
            var chatKey = 'myChatKey';
            sforce.console.chat.onFileTransferCompleted(chatKey, fileSuccess);
        }

        function fileSuccess(result) {
            //Reports status of the file transfer.
            if (result.success == true) {
                alert('File transfer was successful.');
            } else {
                alert('File transfer was not successful.');
            }
        };
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
attachmentId	String	The ID of the object created for the transferred file.

Name	Type	Description
`success`	Boolean	`true` if firing event was successful; `false` if firing event was unsuccessful.

`onNewMessage()`

Registers a function to call when a new message is sent from a customer, agent, or supervisor. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onNewMessage(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	string	The `chatKey` associated with the chat for which to call a function when a new customer message is received.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('There is a new message in this chat: ' + result.content);
        }
        //Get the value for 'myChatKey'from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
        //These values are for example purposes only
        var chatKey = 'myChatKey';
        sforce.console.chat.onNewMessage(chatKey, eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`content`	String	The text of a message in the chat log.
`name`	String	The name of the user who sent the message. This appears exactly as it is displayed in the chat log.
`type`	String	The type of message that was received, such as an Agent or Visitor message.

Name	Type	Description
`timestamp`	Date/Time	The date and time the message was received.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`onTypingUpdate()`

Registers a function to call when the customer's text in the chat window changes. If Sneak Peek is enabled, this function is called whenever the customer edits the text in the chat window. If Sneak Peek is not enabled, this function is called whenever a customer starts or stops typing in the chat window. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.onTypingUpdate(chatKey:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` associated with the chat for which to call a function when a customer begins typing a new message to the agent.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <script type="text/javascript">
        var eventHandler = function (result) {
            alert('There is a new typing update in this chat');
        }
        //Get the value for 'myChatKey'from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
        //These values are for example purposes only
        var chatKey = 'myChatKey';
        sforce.console.chat.onTypingUpdate(chatKey, eventHandler);
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`isTyping`	Boolean	Indicates whether a chat visitor is typing (`true`) or not (`false`).
`sneakPeek`	String	The text the chat visitor is currently typing into their input box in the chat window. This is visible only if Sneak Peek is enabled for the agent.
`success`	Boolean	`true` if firing event was successful; `false` if firing event wasn't successful.

`sendCustomEvent()`

Sends a custom event to the client-side chat window for a chat with a specific chat key. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.sendCustomEvent(chatKey:String, type:String, data:String,
callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` associated with the chat to which to send a custom event.
`type`	String	The name of the custom event you want to send to the chat window.
`data`	String	Additional data you want to send to the chat window along with the custom event.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testSendCustomEvent();">Send Custom Event</a>

    <script type="text/javascript">

        function testSendCustomEvent() {
            //Get the value for 'myChatKey'from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            var type = 'myCustomEventType'
            var data = 'myCustomEventData'
          sforce.console.chat.sendCustomEvent(chatKey, type, data, sendCustomEventSuccess);

        }

        function sendCustomEventSuccess(result) {
            //Report whether sending the custom event was successful
```

```
            if (result.success == true) {
                alert('The customEvent has been sent');
            } else {
                alert('Sending the customEvent was not successful');
            }
        };

    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if sending the custom event was successful; `false` if sending the custom event wasn't successful.

`sendMessage()`

Sends a new chat message from the agent to a chat with a specific chat key. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.sendMessage(chatKey:String, message:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` of the chat where the agent's message is sent.
`message`	String	The message you would like to send from the agent to the customer in a chat.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testSendMessage();">Send Message</a>

    <script type="text/javascript">

        function testSendMessage() {
            //Get the value for 'myChatKey'from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
            //These values are for example purposes only
```

```
            var chatKey = 'myChatKey';
            var text ='This is sample text to send as a message';
            sforce.console.chat.sendMessage(chatKey, text, sendMessageSuccess);
        }

        function sendMessageSuccess(result) {
            //Report whether getting the chat log was successful
            if (result.success == true) {
                alert('Message Sent');
            } else {
                alert('Sending the message was not successful');
            }
        };


    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if sending the message was successful; `false` if sending the message wasn't successful.

`setAgentInput()`

Sets the string of text in the agent's text input area in the chat log of a chat with a specific chat key.Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.setAgentInput(chatKey:String, text:String, callback:Function)
```

Arguments

Name	Type	Description
`chatKey`	String	The `chatKey` associated with the chat for which to set the agent's input text.
`text`	String	The string of text which you want to set into an agent's input.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
```

```
    <a href="#" onClick="testSetAgentInput();">Set Agent Input</a>

    <script type="text/javascript">

        function testSetAgentInput() {
            //Get the value for 'myChatKey'from the
sforce.console.chat.getDetailsByPrimaryTabId() or other chat methods.
            //These values are for example purposes only
            var chatKey = 'myChatKey';
            var text = 'This is example text to set the agent input'
            sforce.console.chat.setAgentInput(chatKey, text, setAgentInputSuccess);
        }

        function setAgentInputSuccess(result) {
            //Report whether setting the agent's input was succesful
            if (result.success == true) {
                alert('The text in the agent input has been updated');
            } else {
                alert('Setting the agent input was not Succesful');
            }
        };


    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if setting the agent's input was successful; `false` if setting the agent's input wasn't successful.

`setAgentState()`

Sets an agent's Chat status, such as Online, Away, or Offline. Available in API version 29.0 or later.

Syntax

```
sforce.console.chat.setAgentState(state:String, (optional)callback:Function)
```

Arguments

Name	Type	Description
`state`	String	Chat status you want to set the agent to—for example, Online, Away, or Offline.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testSetAgentState('Online');return false;">Set Agent Status to
Online</a>
    <script type="text/javascript">
        function testSetAgentState(state) {
            sforce.console.chat.setAgentState(state, function(result) {
                if (result.success) {
                    alert('Agent State Set to Online');
                } else {
                    alert('setAgentState has failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if setting the agent's Chat status was successful; `false` if setting the agent's Chat status wasn't successful.

Methods for Chat Visitors

There are a few methods available that you can use to customize the visitor experience for Chat in a custom Visualforce chat window. These methods apply to Salesforce Classic only.

Important: The legacy chat product is in maintenance-only mode, and we won't continue to build new features. You can continue to use it, but we no longer recommend that you implement new chat channels. Instead, you can modernize your customer communication with Messaging for In-App and Web. Messaging offers many of the chat features that you love plus asynchronous conversations that can be picked back up at any time.

IN THIS SECTION:

chasitor.addCustomEventListener()

Registers a function to call when a custom event is received in the chat window. Available in API version 29.0 or later.

chasitor.getCustomEvents()

Retrieves a list of custom events that have been received in this chat window during this chat session. Available in API version 29.0 or later.

chasitor.sendCustomEvent()

Sends a custom event to the agent console of the agent who is currently chatting with a customer. Available in API version 29.0 or later.

`chasitor.addCustomEventListener()`

Registers a function to call when a custom event is received in the chat window. Available in API version 29.0 or later.

Syntax

```
liveagent.chasitor.addCustomEventListener(type:String, callback:Function)
```

Arguments

Name	Type	Description
`type`	string	The type of custom event you want to listen for.
`callback`	function	JavaScript method called upon completion of the method.

Sample Code–Visualforce

```
<script type="text/javascript">
        function testAddCustomEventListener() {
            //These values are for example purposes only
            var type = 'myCustomEventType'
            liveagent.chasitor.addCustomEventListener(type, customEventReceived)
        }

        function customEventReceived(result) {
            eventType = result.getType();
            eventData = result.getData();
            alert('A custom event of type: ' + eventType + ' has been received with the
following data: ' + eventData);
        };

        testAddCustomEventListener();
</script>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following methods:

Name	Type	Description
`getType`	method	Accesses the type of the custom event that was sent to this chat window. Returns the `type` argument of the `sforce.console.chat.sendCustomEvent()` method used to send this event.
`getData`	method	Accesses the data of the custom event that was sent to this chat window. Returns the `data` argument of the `sforce.console.chat.sendCustomEvent()` method used to send this event.

Name	Type	Description
`getSource`	method	Accesses the source of the custom event that was sent to this chat window—for example, agent or chat visitor.
`getDate`	method	Accesses the date of the custom event that was sent to this chat window. Returns the date and time the event was received.

`chasitor.getCustomEvents()`

Retrieves a list of custom events that have been received in this chat window during this chat session. Available in API version 29.0 or later.

Syntax

```
liveagent.chasitor.getCustomEvents()
```

Sample Code–Visualforce

```
<a href="#" onClick="testGetCustomEvents();">Get Custom Events</a>

<script type="text/javascript">
        function testGetCustomEvents() {
            events = liveagent.chasitor.getCustomEvents();
            eventsCount = events.length;
            alert('The following number of custom events have occurred: ' + eventsCount);

        };
</script>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following methods and properties:

Name	Type	Description
`events`	Array of `event` objects	An array of `event` objects. Each object represents a custom event that has occurred in this chat. Data on each message object can be accessed by the following methods: • `getType()` • `getData()` • `getSource()` • `getDate()`
`getType`	method	Accesses the type of the custom event that was sent to this chat window. Returns the `type` argument of the `sforce.console.chat.sendCustomEvent()` method used to send this event.

Name	Type	Description
`getData`	method	Accesses the data of the custom event that was sent to this chat window. Returns the `data` argument of the `sforce.console.chat.sendCustomEvent()` method used to send this event.
`getSource`	method	Accesses the source of the custom event that was sent to this chat window—for example, agent or chat visitor.
`getDate`	method	Accesses the date of the custom event that was sent to this chat window. Returns the date and time the event was received.

`chasitor.sendCustomEvent()`

Sends a custom event to the agent console of the agent who is currently chatting with a customer. Available in API version 29.0 or later.

Syntax

```
liveagent.chasitor.sendCustomEvent(type:String, data:String)
```

Arguments

Name	Type	Description
`type`	string	The name of the custom event to send to the agent console.
`data`	string	Additional data you want to send to the agent console along with the custom event.

Sample Code–Visualforce

```
<a href="#" onClick="testSendCustomEvent();">Send Custom Event</a>

<script type="text/javascript">
        function testSendCustomEvent() {
            type = 'myCustomEventType';
            data = 'myCustomEventData';
            liveagent.chasitor.sendCustomEvent(type, data);
            alert('The custom event has been sent');
        };
</script>
```

Response

This method returns no responses.

Methods for Omni-Channel

Omni-Channel is a comprehensive customer service solution that lets your call center route any type of incoming work item—including cases, chats, phone calls, or leads—to the most qualified, available agents in your organization. Omni-Channel provides a customizable customer service solution that integrates seamlessly into the Salesforce console and benefits your customers and support agents.

For more information on Omni-Channel, see *Set Up Omni-Channel*.

IN THIS SECTION:

`acceptAgentWork`

Accepts a work item that's assigned to an agent. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.acceptAgentWork(workId:String, (optional) callback:function)
```

Arguments

Name	Type	Description
workId	String	The ID of the work item the agent accepts.
callback	function	JavaScript method to call when an agent accepts the work item associated with the workId.

Sample Code–Visualforce

```
<apex:page>
     <apex:includeScript value="/support/console/63.0/integration.js"/>
     <a href="#" onClick="testAcceptWork();return false;">Accept Assigned Work Item</a>

     <script type="text/javascript">
         function testAcceptWork() {
             //First get the ID of the assigned work item to accept it
             sforce.console.presence.getAgentWorks(function(result) {
                 if (result.success) {
                     var works = JSON.parse(result.works);
                     var work = works[0];
                     if (!work.isEngaged) {
                         //Now that we have the assigned work item ID, we can accept it
                         sforce.console.presence.acceptAgentWork(work.workId,
function(result) {
                             if (result.success) {
                                 alert('Accepted work successfully');
                             } else {
                                 alert('Accept work failed');
                             }
                         });
                     } else {
                         alert('The work item has already been accepted');
                     }
                 }
             });
         }
     </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if accepting the work item was successful; `false` if accepting the work item wasn't successful.

`closeAgentWork`

Changes the status of a work item to "Closed" and removes it from the list of work items in the Omni-Channel widget. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.closeAgentWork(workId:String, (optional) callback:function)
```

Arguments

Name	Type	Description
`workId`	String	The ID of the work item that's closed.
`callback`	function	JavaScript method to call when the work item associated with the `workId` is closed.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testCloseWork();return false;">Close Engaged Work Item</a>
    <script type="text/javascript">
        function testCloseWork() {
            //First get the ID of the engaged work item to close it
            sforce.console.presence.getAgentWorks(function(result) {
                if (result.success) {
                    var works = JSON.parse(result.works);
                    var work = works[0];
                    if (work.isEngaged) {
                        //Now that we have the engaged work item ID, we can close it
                     sforce.console.presence.closeAgentWork(work.workId,function(result)
 {
                            if (result.success) {
                                 alert('Closed work successfully');
                            } else {
                                alert('Close work failed');
                            }
                        });
                    } else {
                        alert('The work item should be accepted first');
                    }
                }
            });
```

```
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if closing the work item was successful; `false` if closing the work item wasn't successful.

`declineAgentWork`

Declines a work item that's assigned to an agent. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.declineAgentWork(workId:String, (optional) declineReason:String,
 (optional) callback:function)
```

Arguments

Name	Type	Description
`workId`	String	The ID of the work item that the agent declines.
`declineReason`	String	The provided reason for why the agent declined the work request.
`callback`	function	JavaScript method to call when an agent declines the work item associated with the `workId`.

Sample Code–Visualforce

```
<apex:page >
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testDeclineWork();return false;">Decline Assigned Work Item</a>

    <script type="text/javascript">
        function testDeclineWork() {
            //First, get the ID of the assigned work item to accept it
            sforce.console.presence.getAgentWorks(function(result) {
                if (result.success) {
                    var works = JSON.parse(result.works);
                    var work = works[0];
                   sforce.console.presence.declineAgentWork(work.workId, function(result)
 {
```

```
                            if (result.success) {
                                alert('Declined work successfully');
                            } else {
                                alert('Decline work failed');
                            }
                        });
                    }
                });
            }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if declining the work item was successful; `false` if declining the work item wasn't successful.

`getAgentWorks`

Returns a list of work items that are currently assigned to an agent and open in the agent's workspace. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.getAgentWorks(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method to call when the list of an agent's work items is retrieved.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetWorks();return false;">Get Agent's Current Work Items</a>


    <script type="text/javascript">
        function testGetWorks() {
            //These values are for example purposes only.
            sforce.console.presence.getAgentWorks(function(result) {
                if (result.success) {
```

```
                    alert('Get work items successful');
                    var works = JSON.parse(result.works);
                    alert('First Agent Work ID is: ' +  works[0].workId);
                    alert('Assigned Entity Id of the first Agent Work is: ' +
works[0].workItemId);
                    alert('Is first Agent Work Engaged: ' + works[0].isEngaged);
                } else {
                    alert('Get work items failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if retrieving the agent's work items was successful; `false` if retrieving the agent's work items wasn't successful.
`works`	JSON string of `work` objects	A JSON string of `work` objects that represents the work items assigned to the agent that are open in the agent's workspace.

`work`

The `work` object contains the following properties:

Name	Type	Description
`workItemId`	String	The ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent.
`workId`	String	The ID of a work assignment that's routed to an agent.
`isEngaged`	Boolean	Indicates whether an agent is working on a work item that's been assigned to them (`true`) or not (`false`).

`getAgentWorkload`

In API version 35.0 and later, we can retrieve an agent's currently assigned workload. Use this method for rerouting work to available agents.

Syntax

```
sforce.console.presence.getAgentWorkload(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method to call when the agent's configured capacity and work is retrieved.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetAgentWorkload();return false;">
        Get Agent's configured capacity and current workload
    </a>

    <script type="text/javascript">
        function testGetAgentWorkload() {
            sforce.console.presence.getAgentWorkload(function(result) {
                if (result.success) {
                    alert('Retrieved Agent Configured Capacity and Current Workload
successfully');
                   alert('Agent\'s configured capacity is: ' + result.configuredCapacity);

                    alert('Agent\'s currently assigned workload is: ' +
result.currentWorkload);
                } else {
                    alert('Get Agent Workload failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if retrieving the agent's work items was successful; `false` if retrieving the agent's work items wasn't successful.
`configuredCapacity`	Number	Indicates the agent's configured primary capacity (work that's assigned to the current user) through Presence Configuration.
`currentWorkload`	Number	Indicates the agent's currently assigned primary workload.
`configuredInterruptibleCapacity`	Number	Indicates the agent's configured interruptible capacity (work that's assigned to the current user) through Presence Configuration.
`currentInterruptibleWorkload`	Number	Indicates the agent's currently assigned interruptible workload.

getServicePresenceStatusChannels

Retrieves the service channels that are associated with an Omni-Channel user's current presence status. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.getServicePresenceStatusChannels(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method to call when the channels associated with a presence status are retrieved.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetChannels();return false;">
        Get Channels Associated with a Presence Status
    </a>

    <script type="text/javascript">
        function testGetChannels() {
            //These values are for example purposes only.
            sforce.console.presence.getServicePresenceStatusChannels(function(result) {
                if (result.success) {
                    alert('Retrieved Service Presence Status Channels successfully');
                    var channels = JSON.parse(result.channels);
                    //For example purposes, just retrieve the first channel
                    alert('First channel ID is: ' + channels[0].channelId);
                   alert('First channel developer name is: ' + channels[0].developerName);

                } else {
                    alert('Get Service Presence Status Channels failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if retrieving the current presence status channels was successful; `false` if the retrieving the current presence status channels wasn't successful.
`channels`	JSON string of `channel` objects	Returns the IDs and API names of the channels associated with the presence status.

`getServicePresenceStatusId`

Retrieves an agent's current presence status. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.getServicePresenceStatusId(callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method to call when the agent's presence status is retrieved.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testGetStatusId();return false;">Get Omni-Channel Status ID</a>

    <script type="text/javascript">
        function testGetStatusId() {
            sforce.console.presence.getServicePresenceStatusId(function(result) {
                if (result.success) {
                    alert('Get Status Id successful');
                    alert('Status Id is: ' + result.statusId);
                } else {
                    alert('Get Status Id failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if retrieving the presence status ID was successful; `false` if the retrieving the presence status ID wasn't successful.
`statusName`	String	The name of the agent's current presence status.
`statusApiName`	String	The API name of the agent's current presence status.
`statusId`	String	The ID of the agent's current presence status.

`login`

Logs an agent into Omni-Channel with a specific presence status. You also can use this method to reconnect to Omni-Channel after a connection error. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.login(statusId:String, (optional) callback:function)
```

Arguments

Name	Type	Description
`statusId`	String	The ID of the presence status. Agents must be given access to this presence status through their associated profile or permission set.
`callback`	function	JavaScript method to call when the agent is logged in with the presence status associated with `statusId`.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testLogin('0N5xx00000000081');return false;">Log In to
Omni-Channel</a>

    <script type="text/javascript">
        function testLogin(statusId) {
            //Gets the Salesforce ID of the presence status entity which the current user
 has been assigned through their permission set or profile.
            //These values are for example purposes only.
            sforce.console.presence.login(statusId, function(result) {
                if (result.success) {
                    alert('Login successful');
                } else {
                    alert('Login failed');
                }
            });
        }
```

```
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if the login was successful; `false` if the login wasn't successful.

`logout`

Logs an agent out of Omni-Channel. Available in API versions 32.0 and later.

Syntax

```
sforce.console.presence.logout((optional) callback:function)
```

Arguments

Name	Type	Description
`callback`	function	JavaScript method to call when the agent is logged out of Omni-Channel.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testLogout();return false;">Log out of Omni-Channel</a>

    <script type="text/javascript">
        function testLogout() {
            sforce.console.presence.logout(function(result) {
                if (result.success) {
                    alert('Logout successfully');
                } else {
                    alert('Logout failed');
                }
            });
        }
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if the logout was successful; `false` if the logout wasn't successful.

setServicePresenceStatus

Sets an agent's presence status to a status with a particular ID. In API version 35.0 and later, we log the user into presence if that user is not already logged in, so you don't have to make additional calls. You also can use this method to reconnect to Omni-Channel after a connection error.

Syntax

```
sforce.console.presence.setServicePresenceStatus(statusId:String,
      (optional) callback:function)
```

Arguments

Name	Type	Description
`statusId`	String	The ID of the presence status you want to set the agent to. Agents must be given access to this presence status through their associated profile or permission set.
`callback`	function	JavaScript method to call when the agent's status is changed to the presence status associated with `statusId`.

Sample Code–Visualforce

```
<apex:page>
    <apex:includeScript value="/support/console/63.0/integration.js"/>
    <a href="#" onClick="testSetStatus('0N5xx00000000081');return false;">Set Presence
Status</a>

    <script type="text/javascript">
        function testSetStatus(statusId) {

            //Sets the user's presence status to statusID. Assumes that the user was
assigned this presence status through Setup.
            //These values are for example purposes only
            sforce.console.presence.setServicePresenceStatus(statusId, function(result) {

                if (result.success) {
                    alert('Set status successful');
                    alert('Current statusId is: ' + result.statusId);
                    alert('Channel list attached to this status is: ' + result.channels);
 //printout in console for lists
                } else {
                    alert('Set status failed');
                }
            });
        }
```

```
    </script>
</apex:page>
```

Response

This method is asynchronous so it returns its response in an object in a callback method. The response object contains the following properties:

Name	Type	Description
`success`	Boolean	`true` if setting the agent's status was successful; `false` if setting the agent's status wasn't successful.
`statusName`	String	The name of the agent's current presence status.
`statusApiName`	String	The API name of the agent's current presence status.
`statusId`	String	The ID of the agent's current presence status.
`channels`	JSON string of `channel` objects	Returns the IDs and API names of the channels associated with the presence status.

Methods for Omni-Channel Console Events

JavaScript can be executed when certain types of events occur in a console, such as when a user closes a tab. In addition to the standard methods for console events, there are a few events that are specific to Omni-Channel. These events apply to Salesforce Classic only.

Omni-Channel Console Events

Event	Description	Payload
`sforce.console.ConsoleEvent.PRESENCE.LOGIN_SUCCESS`	Fired when an Omni-Channel user logs into Omni-Channel successfully. Available in API version 32.0 or later.	• `statusId`—the ID of the agent's current presence status.
`sforce.console.ConsoleEvent.PRESENCE.STATUS_CHANGED`	Fired when a user changes his or her presence status. Available in API version 32.0 or later.	• `statusId`—the ID of the agent's current presence status. • `channels`—channelJSON string of channel objects. • `statusName`—the name of the agent's current presence status. • `statusApiName`—the API name of the agent's current presence status.

Event	Description	Payload
`sforce.console.ConsoleEvent.PRESENCE.LOGOUT`	Fired when a user logs out of Salesforce. Available in API version 32.0 or later.	None
`sforce.console.ConsoleEvent.PRESENCE.WORK_ASSIGNED`	Fired when a user is assigned a new work item. Available in API version 32.0 or later.	• `workItemId`—the ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent. • `workId`—the ID of a work assignment that's routed to an agent.
`sforce.console.ConsoleEvent.PRESENCE.WORK_ACCEPTED`	Fired when a user accepts a work assignment, or when a work assignment is automatically accepted. Available in API version 32.0 or later.	• `workItemId`—the ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent. • `workId`—the ID of a work assignment that's routed to an agent.
`sforce.console.ConsoleEvent.PRESENCE.WORK_DECLINED`	Fired when a user declines a work assignment. Available in API version 32.0 or later.	• `workItemId`—the ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent. • `workId`—the ID of a work assignment that's routed to an agent.
`sforce.console.ConsoleEvent.PRESENCE.WORK_CLOSED`	Fired when the status of an AgentWork object is changed to Closed. Available in API version 32.0 or later.	• `workItemId`—the ID of the object that's routed through Omni-Channel. This object becomes a work assignment with a `workId` when it's assigned to an agent. • `workId`—the ID of a work assignment that's routed to an agent.
`sforce.console.ConsoleEvent.PRESENCE.WORKLOAD_CHANGED`	Fired when an agent's workload changes. This includes receiving new work items, declining work items, and closing items in the console. It's also fired when there's a change to an agent's capacity or Presence Configuration or when the agent goes offline in the Omni-Channel widget.	• `ConfiguredCapacity`—the configured capacity for the agent. • `PreviousWorkload`—the agent's workload before the change. • `NewWorkload`—the agent's new workload after the change.

`channel`

The `channel` object contains the following functions:

Name	Type	Description
`channelId`	String	Retrieves the ID of a service channel that's associated with a presence status.
`developerName`	String	Retrieves the developer name of the the service channel that's associated with the `channelId`.

CHAPTER 4 Other Resources

In addition to this guide, there are other resources available for you as you learn how to use the console APIs.

IN THIS SECTION:

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

SEE ALSO:

Salesforce Help: Salesforce Console

Salesforce Help: Glossary

Salesforce Developers: Getting Started with Salesforce Platform

Salesforce University: Training

Firebug Extension for Firefox

Salesforce Extensions for Visual Studio Code

Console API 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.

Convention	Description

< >	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}"/>`
	`  <apex:column value="{!contact.MailingCity}"/>`
	`  <apex:column value="{!contact.Phone}"/>`
	`</apex:pageBlockTable>`

{ }	In descriptions of syntax, braces ({ }) are typed exactly as shown.
	`<apex:page>`
	`  Hello {!$User.FirstName}!`
	`</apex:page>`

[ ]	In descriptions of syntax, anything included in brackets is optional. In the following example, specifying ***value*** is optional: `data_type variable_name [ = value];`

|	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:
	`Set<data_type> set_name`
	`  [= new Set<data_type>();] |`
	`  [= new Set<data_type>{value [, value2. . .] };] |`
	`  ;`

INDEX

C

M