RingCentral Embeddable

None

None

Table of contents

1. Getting started

1.1 Registering your application

Before you begin development, you will first need to register your application via the RingCentral Developer Console. This step will provision your application a unique set of credentials used to identify your application on the network, and comes with the benefits listed below.

Benefits

- Your application will be less likely impacted by users of other instances of Embeddable
- You will have visibility into the analytics of your user's use of your Embeddable application
- You will be able to better customize your instance of Embeddable
- The "FOR DEMO PURPOSES ONLY" banner will be removed from your application

About the "FOR DEMO PURPOSES ONLY" banner

Some users of RingCentral Embeddable may observe a banner that appears above the dialer that reads, "FOR DEMO PURPOSES ONLY." This banner appears when a Developer uses the default client ID and secret that comes bundled with the RingCentral Embeddable library. This default client ID and secret are intended to make trying out Embeddable easy by eliminating the step of having to pre-register an application.

While nothing prevents a developer from using the default client ID and secret, it is strongly recommended that developers register their own application when deploying and using RingCentral Embeddable in a production context.

1.1.1 Steps in registering an application

1. Login to the Developer Console, creating an account as necessary.
2. Click "Register App"
3. Select "REST API App" and click "Next."
4. Under the Auth section:

- Select "3-legged OAuth flow authorization code"
- Select "Client-side web app, e.g. SPA, Javascript"
- Set "OAuth Redirect URI" to:

Latest Embeddable version: `https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/redirect.html`

```
For fixed version: `https://apps.ringcentral.com/integration/ringcentral-embeddable/$VERSION/redirect.html`
```

5. Under the Security section, add the following "Application scopes:"

- `Call Control`
- `Edit Message`
- `Edit Presence`
- `Internal Messages`
- `Read Accounts`
- `Read Call Log`
- `Read Call Recording` (2.x recordings feature)
- `Read Contacts`
- `Read Messages`
- `Read Presence`
- `RingOut`
- `SMS`
- `VoIP Calling`
- `WebSocketSubscription`
- `Edit Extensions` (2.x SMS templates feature)
- `TeamMessaging` (optional)
- `Video` (optional for Meeting feature)

For all other parameters you are free to select whatever values your prefer. Consult the Developer Guide to learn more about app registration.

1.2 Develop your application

Introducing RingCentral Embeddable 2.0

Consider building your Embeddable application on the next version of RingCentral Embeddable to take full advantage of the new features and capabilities it brings.

Upon successfully registering your application you will be provided with a unique client ID to identify your application on the network. Your next step will be to develop your application using this unique client ID in the RingCentral environment.

1.2.1 Initializing your application

To develop your application, you will need to update two of RingCentral Embeddable's configuration parameters. Set the following parameters:

Parameter	Value
`clientId`	*Your client ID*
`appServer`	`https://platform.ringcentral.com`

Example using a `<script>` tag

```
<script>
  (function() {
    var rcs = document.createElement("script");
    var clientId = "YOUR CLIENT ID";
    var appServer = "https://platform.ringcentral.com"
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?clientId="+clientId+"&appServer="+appServer;
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

Example using an iframe

```
<iframe width="300" height="500" id="rc-widget" allow="microphone"
        src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?clientId=your_app_client_id&appServer=https://
platform.ringcentral.com">
</iframe>
```

1.3 Initialization

1.3.1 Installing Embeddable on a web page

RingCentral Embeddable supports two ways for integrating itself onto a webpage. They include:

- via Javascript
- via an iframe

EMBEDDING VIA JAVASCRIPT

Add following the following code to the `<head>` section of a website. This method is recommended for most applications as it optimised page load performance.

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

You may also load RingCentral Embeddable directly as follows:

```
<script src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js">
</script>
```

> **Learn more about the `defer` and `async` attributes when loading scripts directly**

Initializing Embeddable manually

Starting in `v1.8.5`, one can initialize Embeddable manually using the code below. This is beneficial if your application needs to load a lot of Javascript and you need to optimize page load and rendering speeds. Note: this is only available when initializing via Javacript.

```
<script>
  window.RC_EMBEDDABLE_ADAPTER_MANUAL_INIT = true; // enable manual init
</script>
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

Once the script has been loaded, you can trigger the initialization process as follows:

```
window.RCAdapterInit();
```

And you can dispose of Embeddable as well:

```
window.RCAdapterDispose();
```

EMBEDDING VIA AN IFRAME

Some developers may prefer to load Embeddable directly via an iframe for security or performance reasons. To load Embeddable via an iframe, use the following code.

```
<iframe width="300" height="500" id="rc-widget" allow="autoplay; microphone"
        src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html">
</iframe>
```

1.3.2 Loading a specific version of RingCentral Embeddable

Using the latest build

We prefer developers to load the latest version of RingCentral Embeddable. This is the default behavior when you load Embeddable via a CDN and do not reference a specific version number, or if you use the RingCentral Embeddable embed tool to generate your embed code. Using the latest build ensures that your application will automatically receive updates along with any new bug fixes.

Loading older versions

Developers wishing to load a specific and fixed version of RingCentral Embeddable can do so by plugging their desired version into the following URL format:

```
https://apps.ringcentral.com/integration/ringcentral-embeddable/<version number>
```

For example:

- `https://apps.ringcentral.com/integration/ringcentral-embeddable/1.4.1`
- `https://apps.ringcentral.com/integration/ringcentral-embeddable/1.9.3`

Loading a specific version may be considered more stable by some developers as their application will be insulated from new features, or unintended changes that may not be backwards-compatible.

Update your Redirect URI to match

Starting with version `1.2.0`, please note that the redirect uri of Embeddable must be changed to match the version you are loading. For example:

`https://apps.ringcentral.com/integration/ringcentral-embeddable/1.4.1/redirect.html`

EXAMPLE: LOADING SPECIFIC VERSION VIA JAVASCRIPT

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/1.4.1/adapter.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

1.3.3 Hosting RingCentral Embeddable from your own servers

While not recommended, some developers may wish to download the RingCentral Embeddable javascript library and host it from their own servers. If you elect to do this, make sure you also:

- create a custom `redirect.html` page
- host your `redirect.html` file from the same domain as your Embeddable javascript file
- update your redirect Uri to point to your custom `redirect.html` file

2. Configuration

2.1 Configuring RingCentral Embeddable

Listed below are all supported parameters that can be configured for RingCentral Embeddable. For those parameters that warrant greater explanation a link has been provided to content where you can learn more about using the parameter properly.

Parameter	Default	Description
`appServer`	*production*	Sets the environment in which Embeddable will run. See Setting your environment.
`authorizationCode`	*null*	See Alternative auth methods.
`authorizationCodeVerifier`	*null*	See Alternative auth methods.
`clientId`	*null*	Sets the client Id credential of your Embeddable application. Useful in removing the FOR DEMO PURPOSES ONLY banner. See Using a custom client ID.
`clientSecret`	None	*Deprecated*. Sets the client secret of your application when using Authorization code grant type. See Alternative auth methods.
`defaultCallWith`	`browser`	See Calling features.
`defaultDirection`	`right`	Allowed values are "left" and "right". See Embeddable Badge.
`disableMessages`	False	Disable messages feature.
`disableReadText`	False	Disable SMS and read text feature.
`disableCall`	False	Disable call-related features.
`disableCallHistory`	False	Disable call history.
`disableContacts`	False	Disable contacts feature.
`disableMeeting`	False	Disable meeting feature.
`disableGlip`	True	Before we start to use Glip API, need to add `Glip` or `Team Messaging` permission to your app in RingCentral Developer website. Also, for testing with a sandbox user, user needs to first login to https://app.devtest.ringcentral.com
`disableMinimize`	False	By default, we provide Minimize button at app header to minimize the widget.
`disconnectInactiveWebphone`	False	See Working with multiple tabs.
`enableAnalytics`	False	See Custom analytics.
`enableNoiseReductionSetting`	False	See Noise reduction.
`enablePopup`	False	See Customize pop-up window.
`enableRingtoneSettings`	False	For when call is ringing, app will play default ringtone. But we also support to customize ringtone. By enabled, user can get ringtone settings at settings page. Supported after v1.6.3
`enableSMSTemplate`	False	See SMS templates.
`jwt`	None	See Alternative auth methods.
`multipleTabsSupport`	False	See Working with multiple tabs.
`newAdapterUI`	False	See Embeddable Badge.
`popupPageUri`	*null*	See Customize pop-up window.
`prefix`	*null*	See Customize prefix.
`redirectUri`	*null*	See Customize redirectUri.
`showSignUpButton`	False	

Parameter	Default	Description
`stylesUri`	*null*	See Customize look and feel through CSS.
`userAgent`	*null*	See Customize X-User-Agent.
`enableAudioInitPrompt`	*null*	See Audio.
`enableLoadMoreCalls`	*null*	Support to load old call history (more than 7 days). Supported from v2.1.0

2.2 Setting RingCentral Embeddable configuration parameters

RingCentral Embeddable supports a number of different configuration parameters to modify the behavior of the library is key ways. Each parameter is set in one of two ways.

Via script tag's `src` attribute

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?parameterName=VALUE";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

Via iframe's `href` attribute

```
<iframe width="300" height="500" id="rc-widget" allow="microphone"
  src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?parameterName=VALUE">
</iframe>
```

2.3 Advanced parameters

2.3.1 Customize X-User-Agent

We provide default `X-User-Agent` header as `RingCentralEmbeddable/0.2.0 RCJSSDK/3.1.3` in RingCentral API request for SDK usage analysis in backend. In this API, developers can also provide their desired User Agent into widget.

After that widget will change `X-User-Agent` header into `TestAPP/1.0.0 RingCentralEmbeddable/0.2.0 RCJSSDK/3.1.3` when send request to RingCentral Server.

2.3.2 Using a custom client ID with RingCentral Embeddable

Upon successfully registering your application you will be provided with a unique client ID to identify your application on the network.

Develop your application in sandbox

Before you can deploy your newly created application to production and remove the FOR DEMO PURPOSES ONLY banner, you will need to point RingCentral Embeddable at RingCentral's Developer sandbox environment.

Javascript iframe

```
<script>
  (function() {
    var rcs = document.createElement("script");
    var clientId = "YOUR CLIENT ID";
    var appServer = "https://platform.devtest.ringcentral.com"
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?clientId="+clientId+"&=appServer="+appServer;
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone"
    src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?clientId=your_app_client_id&appServer=https://
platform.devtest.ringcentral.com">
</iframe>
```

`appKey` and `appSecret` have been renamed

Starting in version `v1.4.0`, `appKey` has been renamed to `clientId` and `appSecret` has been renamed to `clientSecret`.

Graduate your app

Use in production

Currently, new created RingCentral app will go to production directly. You don't need this section now.

Sandbox is used by developers when building and testing applications. In order to use the app in production, developers must "graduate" their app. For RingCentral Embeddable apps, the following should be done in sandbox to quality for graduation:

- Send more than 5 SMS messages
- Send more than 5 internal messages (SMS message to extension number in current account)
- Read more than 5 unread inbound messages
- Update presence more than 5 times in setting page
- Go to Contacts page, and click refresh button more than 5 times
- Login and logout more than 5 times
- Make 5 outbound web phone (Browser based) calls
- Make 5 inbound web phone (Browser based) calls
- Make 5 Ringout calls
- Control(end/hold) Ringout call in widget more than 5 times in widget

Having difficulty graduating your app?

Please create developer support ticket if you experience issues during the app graduation process.

Update your client ID and server URL

Once you have successfully graduated your app, be sure to update the `clientID` and `serverUrl` your instance of Embeddable points to in order to run successfully in production.

2.3.3 Customize look and feel through CSS

This is a online demo built with Game of Thrones Styles.

Style file is defined here:

`https://embbnux.github.io/ringcentral-web-widget-styles/GameofThrones/styles.css`

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?stylesUri=https://embbnux.github.io/ringcentral-web-widget-styles/GameofThrones/styles.css";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?stylesUri=https://embbnux.github.io/ringcentral-web-widget-styles/GameofThrones/styles.css">
</iframe>
```

See a live demo

2.3.4 Customize Redirect Uri

In authorization code flow, it will require a valid redirect uri that developer set in developers account. This app offers a default redirect uri option that you can use, `https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/redirect.html`. And it also supports to config redirect uri.

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?redirectUri=your_redirect_uri";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
redirectUri=your_redirect_uri">
</iframe>
```

Hosting a custom `redirect.html` file

In your redirect page, you need to add following code to pass auth callback params to this app.

```
<script>
  // Important: the origin is used for postMessage
  // Set "origin" to the same domain as the Embeddable library
  var origin = 'https://apps.ringcentral.com';
  if (window.opener) {
    // pass callbackUri to widget
    window.opener.postMessage({
      callbackUri: window.location.href,
    }, origin);
    window.close(); // close the login popup window
  }
</script>
```

> ⚠️ **Be sure to host your redirect.html and Embeddable library from the same domain**
>
> To comply with browser security policies meant to prevent XSS vulnerabilities, both your `redirect.html` file and the RingCentral Embeddable javascript file must be hosted from the same domain. If not, users will be unable to authenticate.

2.3.5 Customize Prefix

We provide default prefix `rc-widget` in the widget. It will used at iframe id prefix and storage key prefix, such as `rc-widget-adapter-frame` and `rc-widget-GlobalStorage-rateLimiterTimestamp`.

Some developers wants to customize the prefix, so the widget can support to have different user storage data. We provide prefix param to support this feature:

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?prefix=your_prefix";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
prefix=your_prefix">
</iframe>
```

After that the widget iframe id will changed to `your_prefix-adapter-frame`. And user data will be storaged at `you_prefix` namespace.

For implicit grant flow, we use cookie to refresh the token, so it don't support different accounts in same browser in different tabs. If you want to support different accounts in different tabs or domains in same browser, you need to use authrization code flow.

2.3.6 Interact with Calling Settings

In RingCentral Embeddable widget, we provide 4 calling options in `Calling Setting` page.

- Browser - make and receive calls using your computer's microphone and speaker based on browser
- RingCentral App - make and receive calls using your `RingCentral` desktop app
- RingCentral Phone - make and receive calls using your `RingCentral Phone` desktop app
- RingOut

For `RingOut`, You can get full document here Ringout mode. Users also need to set `My Location` phone number. So when user creates a call, RingCentral will first call user's location phone number, then call correspondent's phone number. If user enables `Prompt me to dial 1 before connecting the call`, RingCentral will only call correspondent's phone number after user dials 1. Please refer to here for more detailed information.

For `RingCentral App`, `RingCentral Phone`, `RingOut`, calls are on other devices, the widget can get call event and information from APIs. And it is recommended to enable active call control, so user can also control the call in widget.

Default option

To set default `callWith` option:

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?defaultCallWith=browser";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
defaultCallWith=browser">
</iframe>
```

There are 4 options for `defaultCallWith`:

- browser
- jupiter
- softphone
- ringout

They are short names of `Browser`, `RingCentral App`, `RingCentral Phone`, `RingOut`.

Calling settings updated event

Event fired when user changed `call with` option in calling settings page:

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-calling-settings-notify':
        // get calling setting in here
        console.log(data);
        // {
        //   type: 'rc-calling-settings-notify',
        //   callWith: 'ringout',
        //   fromNumbers: [], // show after enableFromNumberSetting flag set
        //   myLocation: '+11111111', // show after showMyLocationNumbers flag set
        //   myLocationNumbers: [], // show after showMyLocationNumbers flag set
        // }
        break;
      default:
        break;
    }
```

```
  }
});
```

Enable call from number setting

In widget, user can also select `From` number when make a browser call. For developers who also want to set `From` number programmatically, we need to enable from number settings:

Javascript iframe

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?enableFromNumberSetting=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
enableFromNumberSetting=1">
</iframe>
```

After enabled, we can receive `From` number list in calling settings updated event when `callWith` is `browser`.

Show my location numbers

1.8.1

In `RingOut` mode, user need to set `My Location` number to receive first-leg call. For developers who also want to get user's known location numbers programmatically, we need to set `showMyLocationNumbers` flag firstly:

Javascript iframe

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?showMyLocationNumbers=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
showMyLocationNumbers=1">
</iframe>
```

After enabled, we can receive `myLocation` and `myLocationNumbers` in calling settings updated event.

Update Calling settings

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-calling-settings-update',
  callWith: 'softphone',
  // myLocation: '+1111111111', // required for ringout
  // ringoutPrompt: true, // required for ringout
  // fromNumber: '+1111111111', // set from number when callWith is browser
}, '*');
```

For `fromNumber`, the number should be from `fromNumbers` list in calling settings event, or `anonymous` for `Blocked` from number.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-calling-settings-update',
  fromNumber: 'anonymous',
}, '*');
```

2.3.7 Multiple Brands

RingCentral works with a number of carriers and partners to deliver a cutting edge white labeled Cloud Communications service directly to their respective customers. This guide will show you how to create RingCentral Embeddable app for other brands.

AT&T Office@Hand

ADAPTER JS WAY

- Use `adapter.att.js` to instead of `adapter.js`
- Set `appServer` to `https://platform.ringcentral.biz`

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.att.js?appServer=https://platform.ringcentral.biz";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

IFRAME WAY

Add `brand=att` and `appServer` in src query parameter:

```
<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
brand=att&appServer=https://platform.ringcentral.biz">
</iframe>
```

BT Cloud Work

Use `adapter.bt.js` to instead of `adapter.js`:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.bt.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

IFRAME WAY

Add `brand=bt` in src query parameter:

```
<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
brand=bt">
</iframe>
```

TELUS

Use `adapter.telus.js` to instead of `adapter.js`:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.telus.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

IFRAME WAY

Add `brand=telus` in src query parameter:

```
<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
brand=telus">
</iframe>
```

Atos Unify Office

From v1.8.0

Use `adapter.atos.js` to instead of `adapter.js`:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.atos.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

IFRAME WAY

Add `brand=atos` in src query parameter:

```
<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
brand=atos">
</iframe>
```

Avaya Cloud Office

From v1.8.0

Use `adapter.avaya.js` to instead of `adapter.js`:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.avaya.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

IFRAME WAY

Add `brand=avaya` in src query parameter:

```
<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
brand=avaya">
</iframe>
```

Rainbow Office

From v1.8.0

Use `adapter.rainbow.js` to instead of `adapter.js`:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.rainbow.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

IFRAME WAY

Add `brand=rainbow` in src query parameter:

```
<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
brand=rainbow">
</iframe>
```

Other Brands

For other brands, we are still customizing styles for them. Those users can use with default brand.

ADAPTER JS WAY

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

IFRAME WAY

```
<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html">
</iframe>
```

2.3.8 Noise reduction

1.10.0

Currently in beta.

Noise reduction is a self-descriptive feature that when enabled, filters out background noise present in the user's environment to create a clearer, easier-to-hear audio stream for people on the other end of a call. Noise reduction is currently only supported within the Google Chrome and Microsoft Edge browsers.

How to enable noise reduction

> **Noise reduction is only supported when loading Embeddable from `apps.ringcentral.com`**
>
> Noise reduction is not supported if you are loading the Embeddable library from a Github domain. The embeddable library must be loaded from `https://apps.ringcentral.com` domain. Please check your source code and migrate to a more new recent build hosted at ringcentral.com."

1.10.0

At version `1.10.x`, noise reduction is disabled by default. To enable noise reduction, developer need to pass `enableNoiseReductionSetting` at query parameters.

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?enableNoiseReductionSetting=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone"
  src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?enableNoiseReductionSetting=1">
</iframe>
```

2.0.0

From `v2.0.0`, noise reduction feature is enabled by default for supported browsers. User can disable it manually in settings page. If you want to disable and remove the feature, please check How to remove noise reduction feature.

How to remove noise reduction feature

2.0.0

Noise reduction settings will be showed in settings page automatically from `v2.0.0`. If you don't want to use noise reduction feature, you can remove it by setting `disableNoiseReduction` at query parameters.

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?disableNoiseReduction=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone"
  src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?disableNoiseReduction=1">
</iframe>
```

2.3.9 Popup the widget in a standalone window

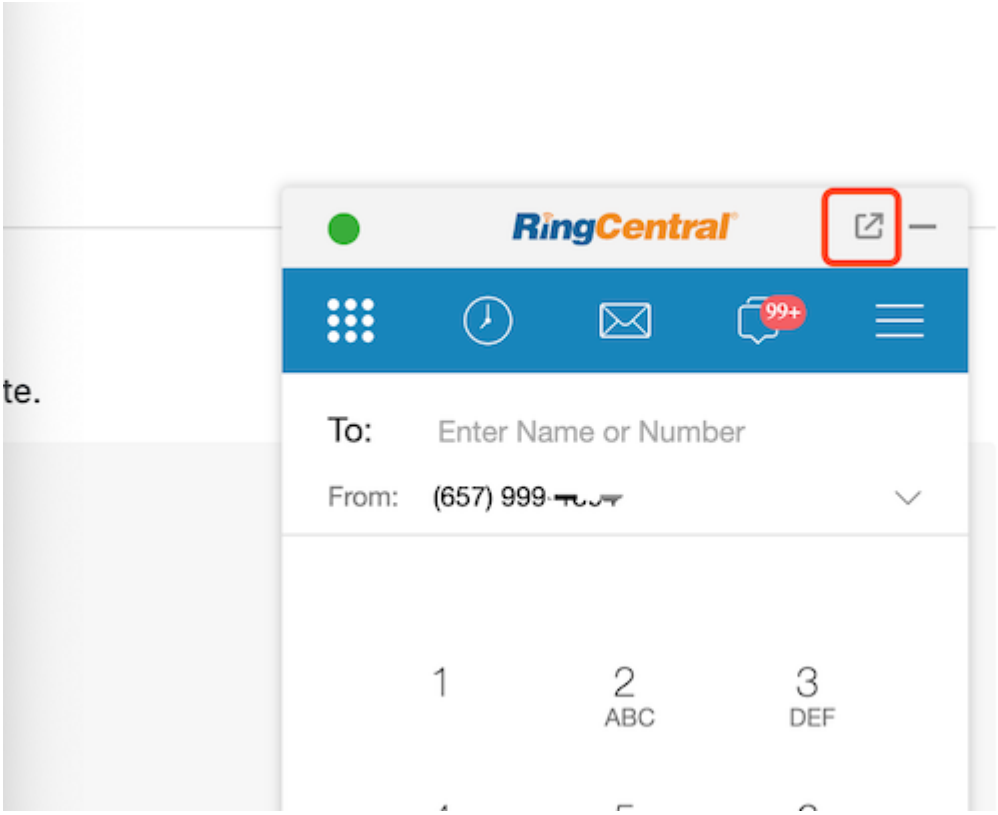
Support to open the widget in a popup window, so the widget is opened at a standalone window. User can close web page that embed the widget, and call will not be ended, and still active at popup window.

For `Chrome (from 117)`, Safari and Firefox, iframe inside different domain is isolated. As default popup window's domain is different as the page embedded, need to host the popup window in same domain origin as the page embedded to have this feature work.

Javascript

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?enablePopup=1&multipleTabsSupport=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

After enabled, user will get a popup button at header:



The feature is based on Webphone Multiple Tabs solution 1, so `multipleTabsSupport` need to be enabled. If you have enable `disconnectInactiveWebphone`, please remove it. Before user open popup window, web phone connection is built at first opened tab. After user open popup window, web phone connection is built at popup window.

To check if popup window opened

From v1.8.0:

```
RCAdapter.isWindowPoppedUp().then((opened) => {...})
```

Known issues

- App can't make a opened popup window into desktop top (Browser limitation)
- App will send Web phone call session notification at every tabs
- User need to focus at popup window when start or answer a call at popup window for microphone permission at Firefox

Host the popup window

For some reason, developers need to host the popup HTML file by themselves. For example, if developer want to add Third Party service register and response into the widget, it is required to host the popup HTML file in your domain, and add your script inside the HTML file. It can be also used for resolve cross-origin domain issue.

In this case, we can config the popup button to open your own popup HTML file URI:

Javascript

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?enablePopup=1&popupPageUri=your_popup_html_file_uri";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

The HTML file need to be based on code of this file. Then update the `adapter.js` src into absolute address in the file:

```
<script src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js"></script>
```

Then add your own script in the file.

2.3.10 Multiple Tabs

For the Embeddable widget, it supports to run in multiple tabs, and will share the same storage and login status. But widgets in different tabs are still different instances.

When calling mode is set into Browser, widgets will create web phone connection in every widget instance. In our server-side, we have limitation of 5 phone connection. So when user selects `Browser` to make call, we only support to open tabs that no more than 5.

Option 1: Have only connection in first connected tab

To resolve 5 tab limitation issue for multiple tabs (more than 5), we have this option to make only a web phone connection in multiple tabs.

CORE IDEA

1. Web phone connection is only connected in first connected tab.
2. When user has a call in second tab or third tab etc, voice transmission is happened in first tab. Second tab only has web phone UI.
3. When user controls call in second tab, control command sent to first tab to execute.
4. When user closes first tab, second tab becomes first opened tab. Web phone will be connected in this tab.
5. Web phone states are shared with local storage between different tabs.
6. Use localStorage as message channel between different tabs.

> **This feature is in beta, we need more tests and feedback about it. It only works after v1.5.0.**

Javascript iframe

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?multipleTabsSupport=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
multipleTabsSupport=1">
</iframe>
```

> **Known issues**
>
> - For Safari and Firefox, users need to go back to first opened tab to allow microphone permission for every call.
> - For Chrome, user need to go back to first opened tab to allow microphone permission if user hasn't allowed microphone permission.
> - Web phone call session notification happens at all tabs with the widget.
> - Web phone call muted event does not work at no web phone connection tabs.
> - For Chrome and Firefox, browser may throttle or unload inactive (5 mins) tabs to make this feature broken.

Option 2: disconnect inactive web phone

For 5 tab limitation, we support to disconnect web phone connection in inactive tabs. So user can open more than 5 tabs, and not more than 5 active tabs.

CORE IDEA:

1. When user goes to new tab and web phone is connected, web phone connection in inactive tabs will be disconnected.
2. When user goes back to inactive tab, the tab became active and widget will reconnect web phone connection.

3. When user has active calls in inactive tabs, web phone connection in inactive tabs will be kept unless all calls ended.
4. User can control calls from inactive tabs by Call Control RESTful API in active tab. And can switch calls into current active tab.

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?disconnectInactiveWebphone=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
disconnectInactiveWebphone=1">
</iframe>
```

Known issues

- App will show connecting badge a while after user change active tab
- Performance issue when user change active tab fast
- At Firefox, app can't disconnect web phone successfully at active page unloaded. So it maybe show too many connection error.

2.3.11 Setting your environment

RingCentral supports two different environments in which applications can run. These two environments are:

- **Production**. This is the primary environment for normal RingCentral operations.
- **Sandbox**. This is an environment set aside exclusively for developers to build and test applications before making them available in production.

By default, RingCentral Embeddable's `appServer` configuration parameter points to production.

Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    var clientId = "YOUR CLIENT ID";
    var appServer = "https://platform.devtest.ringcentral.com"
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?clientId="+clientId+"&=appServer="+appServer;
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone"
        src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?clientId=your_app_client_id&appServer=https://
platform.devtest.ringcentral.com">
</iframe>
```

`appKey` and `appSecret` have been renamed

Starting in version `v1.4.0`, `appKey` has been renamed to `clientId` and `appSecret` has been renamed to `clientSecret`.

2.3.12 SMS template

2.0.0

The SMS template feature is supported starting in version 2.0.0. This feature allows users to manage and utilize a set of shared pre-written messages in the SMS messages sent via RingCentral Embeddable. When enabled, users can:

- Select and apply an templated message in an SMS they are writing
- Create new SMS templates
- Access templates create by coworkers or administrators

Enable SMS template

First, you need to add `EditExtensions` permission into your RingCentral app in RingCentral developer portal.

Then, you need to enable the SMS template feature in the widget. To enable it, you need to set `enableSMSTemplate` flag into the widget's URI.

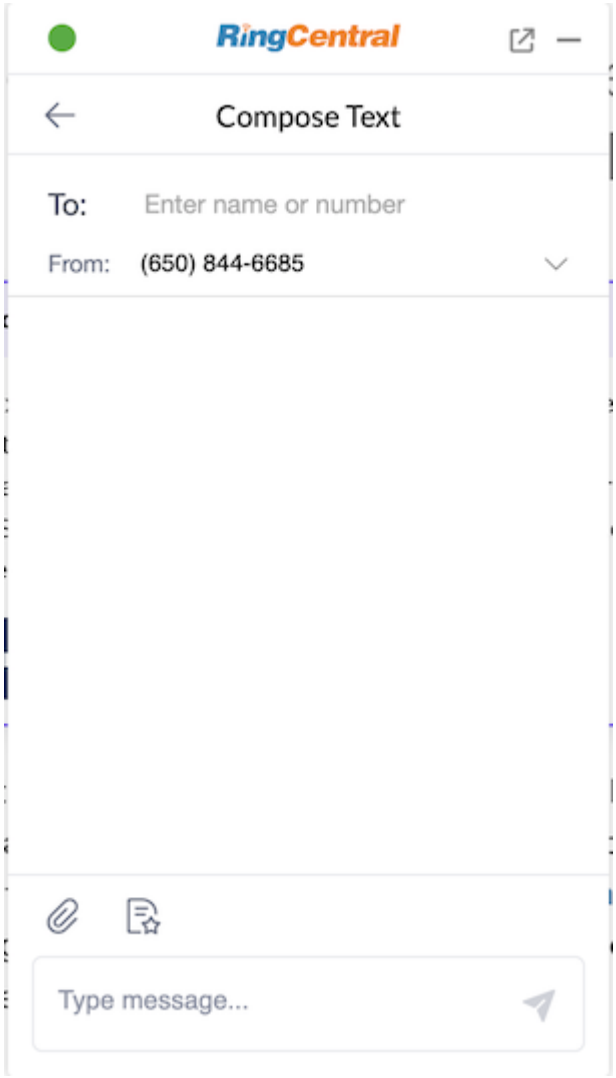
Javascript **iframe**

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?enableSMSTemplate=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
enableSMSTemplate=1">
</iframe>
```

Use SMS template

After enabled, user should be able to see the SMS template tab SMS text input toolbar.



Import SMS template

The widget provides a API to import SMS template into the widget. You can use the following code to import SMS template into the widget.

Adapter JS Javascript

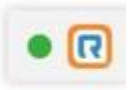
```
RCAdapter.createSMSTemplate('Template name', 'Template text');

document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-message-request',
  requestId: Date.now().toString(),
  path: '/create-sms-template',
  body: {
    displayName: 'Template name',
    text: 'Template text'
  },
}, '*');
```

2.3.13 Embeddable badge

In adapter JS way, our codes will generate a `RingCentral Badge` in web page by default:

In latest version, we implement a new dock UI:



Changing the location of the badge

Use the `defaultDirection` configuration parameter to dock the badge either the `left` or `right` of the window.

2.3.14 Audio

When the calling setting is configured to "Browser", the widget utilizes the browser's default audio and microphone devices for capturing audio and playing ringtones or call voices. Users can also change the audio device via audio settings page.

Audio output

Due to browser limitations, the widget can only play audio after user interacts with it. Therefore, an audio initialization is required to enable ringtone playback for incoming calls. Users can initialize audio by interacting with any button within the widget.

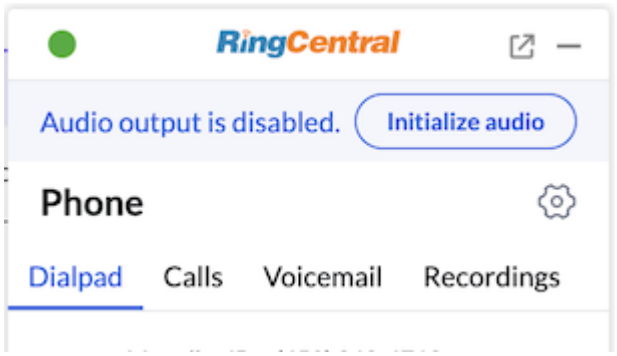
And developer can enable `Audio Initialization` banner to remind user to initialize audio. The banner will be displayed at the top of the widget.

2.1.0

Javascript iframe

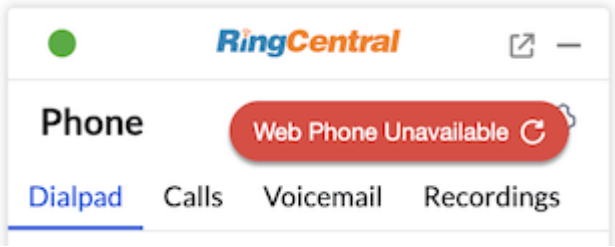
```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?enableAudioInitPrompt=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
enableAudioInitPrompt=1">
</iframe>
```



Audio input

Access to a microphone device is essential for the widget to capture audio. If the widget cannot access the microphone, it will display a `Web Phone Unavailable` badge:



There are some reasons the widget might fail to access the microphone:

USER PERMISSION

The user must grant permission for the widget to access the microphone. If permission is denied, the widget will be unable to capture audio. When the user clicks on the `Web Phone Unavailable` badge, the browser will prompt a dialog requesting microphone access.

NO MICROPHONE DEVICE

If the user's device does not have a microphone, the widget will be unable to capture audio. Users can connect an external microphone to resolve this issue.

HTTP PROTOCOL

Microphone access is only permitted over secure connections (HTTPS). If the website is not using HTTPS, the browser will prevent the widget from accessing the microphone.

PERMISSIONS-POLICY

For security reasons, some websites may implement a Permissions-Policy header that restricts microphone usage. If such a header is present, the widget will only be able to access the microphone if the policy explicitly permits it. The header should be formatted as follows:

```
Permissions-Policy: microphone=("https://apps.ringcentral.com" "self")
```

3. Developer guide

3.1 RingCentral Embeddable events

RingCentral Embeddable emits a number of events that a developer can subscribe to in order to integrate more deeply with the library. Subscribing to these events is done via the postMessage API.

3.1.1 Active call event

This event is fired for all calling modes, even when the call is on another device within the same RingCentral account. Get all active calls in current RingCentral logged user (extension) via a message event.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-active-call-notify':
        // get call on active call updated event
        console.log(data.call);
        break;
      default:
        break;
    }
  }
});
```

> **Working with multiple instances of Embeddable**
>
> If user opens multiple tabs, the event will be fired in every tab. `disableInactiveTabCallEvent` is a option that makes widget only fire active call event in last active tab. Just add `disableInactiveTabCallEvent=1` in widget adapter js uri or iframe src.

3.1.2 Dialer status event

Before we use the API to open the dialer, we need to check dialer status to make sure it is ready. This event fires whenever the status of the dialer changes.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-dialer-status-notify':
        // get dialer status from widget
        console.log('rc-dialer-status-notify:', data.ready);
        break;
      default:
        break;
    }
  }
});
```

3.1.3 Login popup event

Embeddable will open a popup window in order to login a user when that user clicks the login button. For some reason, you may want to popup window by yourself. So you can use login popup event to get login URI for open login window.

For enabling this event, set the `disableLoginPopup=1` configuration parameter.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-login-popup-notify':
        // get login oAuthUri from widget
        console.log('rc-login-popup-notify:', data.oAuthUri);
        //  window.open(data.oAuthUri); // open oauth uri to login
        break;
      default:
```

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

This event also allows you to intercept the RingCentral authorization code if you so choose to faciliate authorization, which you can use to complete the authorization process for Embeddable.

3.1.4 Login status event

You can receive changes to the user's current login status via this event, allowing you reinitiate the login process if the user is loged out, or to perform other login-related operations.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-login-status-notify':
        // get login status from widget
        console.log('rc-login-status-notify:', data.loggedIn, data.loginNumber);
        break;
      default:
        break;
    }
  }
});
```

3.1.5 Meeting status event

Get meeting status and permission:

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-meeting-status-notify':
        // get meeting status and permission from widget
        console.log('rc-meeting-status-notify:', data.ready, data.permission);
        break;
      default:
        break;
    }
  }
});
```

3.1.6 Message event

Get all message created or updated events via Embeddable. These message events include events relating to:

- SMS messages sent/received
- Voicemails received
- Fax messages sent/received

This does not include Team chat messaging events.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-message-updated-notify':
        // get message from widget event
        console.log('rc-message-updated-notify:', data.message);
        break;
      default:
        break;
    }
  }
});
```

New inbound messages

Get new inbound message event from Embeddable.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-inbound-message-notify':
        // get new inbound message from widget event
        console.log('rc-inbound-message-notify:', data.message);
        break;
      default:
        break;
    }
  }
});
```

3.1.7 Presence sync event

Subscribe to presence change events for the currently logged in user.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-adapter-syncPresence':
        // get dndStatus, telephonyStatus, userStatus defined here
        // https://developers.ringcentral.com/api-reference/Extension-Presence-Event
        console.log(data);
        break;
      default:
        break;
    }
  }
});
```

You can modify a user's presence via Embeddable's API

3.1.8 Region settings event

Subscribe to any changes to a user's region settings.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-region-settings-notify':
        // get region settings from widget
        console.log('rc-region-settings-notify:', data);
        break;
      default:
        break;
    }
  }
});
```

3.1.9 RingOut call event

This event is fired when calling mode is set to `My RingCentral Phone` or `Custom Phone`. Get the RingOut call event via message event:

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-ringout-call-notify':
        // get call on active call updated event
        console.log(data.call);
        break;
      default:
        break;
    }
  }
});
```

Learn more about RingOut

3.1.10 Route changed event

Get Current page route from widget

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-route-changed-notify':
        // get current page route from widget
        console.log('rc-route-changed-notify:', data.path);
        break;
      default:
        break;
    }
  }
});
```

3.1.11 Telephony Session Event

Telephony Session is active call data from new Call Control API. In telephony session, we can get full state of caller and callee. We can use Telephony Session event instead of Active Call event.

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-telephony-session-notify':
        // get telehony session on telephony session event
        console.log(data.telephonySession);
        break;
      default:
        break;
    }
  }
});
```

> ⚠ **Working with multiple instances of Embeddable**
>
> If user opens multiple tabs, the event will be fired in every tab. From `v1.10.1`, `disableInactiveTabCallEvent` is a option that makes widget only fire this event in last active tab. Just add `disableInactiveTabCallEvent=1` in widget adapter js uri or iframe src.

3.1.12 Web phone call event

These events are only fired when calling mode is set to `Browser` and the user has received a call via Embedddable.

Get web phone (Browser) call info from web phone call event:

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-call-ring-notify':
        // get call when user gets a ringing call
        console.log(data.call);
        break;
      case 'rc-call-init-notify':
        // get call when user creates a call from dial
        console.log(data.call);
        break;
      case 'rc-call-start-notify':
        // get call when a incoming call is accepted or a outbound call is connected
        console.log(data.call);
        break;
      case 'rc-call-hold-notify':
        // get call when user holds a call
        console.log(data.call);
        break;
      case 'rc-call-resume-notify':
        // get call when user unholds call
        console.log(data.call);
        break;
      case 'rc-call-end-notify':
        // get call on call end event
        console.log(data.call);
        break;
      case 'rc-call-mute-notify':
        // get call on call muted or unmuted event
```

```
        console.log(data.call);
        break;
      default:
        break;
    }
  }
});
```

Call event types

Event	Trigger
`rc-call-ring-notify`	fired when user gets a ringing incoming call
`rc-call-init-notify`	fired when user create a call from dial pad
`rc-call-start-notify`	fired when user accepts a ringing call or a outbound call is connected
`rc-call-hold-notify`	fired when user holds a call
`rc-call-resume-notify`	fired when user unholds a call
`rc-call-end-notify`	fired when call is ended
`rc-call-mute-notify`	fired when call is muted or unmuted

VOIP call vs physical phone calls

When user creates a call to a physical phone number, `rc-call-start-notify` is fired when callee accepts call. When user creates a call to a VOIP phone number (such as bettween RingCentral account), `rc-call-start-notify` is fired when outbound call is ringing in callee side.

3.1.13 Web phone connection status event

1.8.3

Embeddable's web phone (browser-based calling) works only after having successfully connected with a SIP server. To detect when the phone is connected:

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-webphone-connection-status-notify':
        // get call on active call updated event
        console.log(data.connectionStatus); // connectionStatus-connected, connectionStatus-disconnected
        break;
      default:
        break;
    }
  }
});
```

3.1.14 Web phone sessions sync event

1.8.3

To get current active web phone calls send the sync trigger to Embeddable.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-webphone-sessions-sync',
}, '*');
```

Only send this trigger *after* getting web phone connected event.

Receive active web phone calls via message event:

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-webphone-sessions-sync':
        console.log(data.calls);
        break;
      default:
        break;
    }
  }
});
```

3.2 Embeddable API for controlling and manipulating the UI

Embeddable provides an API that allows developers to control the UI and flow of the application via the embedded CTI. It is based on the postMessage API.

3.2.1 Active call control

1.x

Following APIs need to work with Web phone call event to get `callId`.

Answer a ringing call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'answer',
  callId: `call id`
}, '*');
// callId comes from web phone call event

// answer the current ringing call, call id default is current ringing call id.
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'answer',
}, '*');
```

Reject a ringing call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'reject',
  callId: `call id`
}, '*');
```

To voicemail a ringing call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'toVoicemail',
  callId: `call id`
}, '*');
```

Forward a ringing call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'forward',
  callId: `call id`,
  options: {
    forwardNumber: 'forward_number'
  }
}, '*');
```

Hangup a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'hangup',
  callId: `call id`
}, '*');

// hangup current active call
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'hangup',
}, '*');
```

Hold a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'hold',
  callId: `call id`
}, '*');
```

Unhold a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'unhold',
  callId: `call id`
}, '*');
```

Transfer a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'transfer',
  callId: `call id`,
  options: {
    transferNumber: 'transfer_number'
  }
}, '*');
```

Record a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'startRecord',
  callId: `call id`
}, '*');
```

This will only work after the call has already started (Inbound call accepted/Oubound call connected)

Stop record a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'stopRecord',
  callId: `call id`
}, '*');
```

Mute a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'mute',
  callId: `call id`
}, '*');
```

Unmute a call

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
  callAction: 'unmute',
  callId: `call id`
}, '*');
```

Close/open current ringing page

2.2.1

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-control-call',
```

```
  callAction: 'toggleRingingDialog',
}, '*');

## Embeddable widget

### Show custom alert message

<span class="mdx-badge"><span class="mdx-badge__icon">[:material-tag-outline:](https://github.com/ringcentral/ringcentral-embeddable/releases 'Minimum
version')</span><span class="mdx-badge__text">[1.8.6](https://github.com/ringcentral/ringcentral-embeddable/releases)</span></span>

```js
const requestId = Date.now().toString();
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-message-request',
  requestId: requestId,
  path: '/custom-alert-message',
  alertMessage: 'Test info message',
  alertLevel: 'info',
  ttl: 5000 //5000ms => 5s
}, '*');
```

Note: `alertLevel` can be `info`, `warning` or `danger`.

2.0.0

```
const alertId = await RCAdapter.alertMessage({
  message: 'Test info message',
  level: 'info',
  ttl: 5000 //5000ms => 5s, 0 for infinite
});
// Dismiss the alert message
RCAdapter.dismissMessage(alertId); // dismiss the alert message
// Dismiss all alert messages
RCAdapter.dismissMessage();
```

2.1.0

Alert with details

```
const alertId = await RCAdapter.alertMessage({
  message: 'Can not log phone call',
  ttl: 0,
  level: 'danger',
  details: [{
    title: 'Description',
    items: [{
      id: '1',
      type: 'text',
      text: 'This is a longer more descriptive explanation of the problem that can expand over multiple lines.'
    }],
  }, {
    title: 'Error from CRM',
    items: [{
      id: '1',
      type: 'text',
      text: 'OAU-1973 Contact not found'
    }],
  }, {
    title: 'Actions',
    items: [{
      id: '1',
      type: 'link',
      text: 'Need help? Ask out community.',
      href: 'https://community.ringcentral.com'
    }, {
      id: '2',
      type: 'link', // when not href set, it will send a event with [button event](./sms-toolbar-button.md)
      text: 'Resolve logging conflict',
    }]
  }]
})
```

Minimize/Hide/Remove the widget

Only for Adapter JS way:

Minimize:

```
RCAdapter.setMinimized(true);
// RCAdapter.setMinimized(false); // maximize
```

You can also disable `Minimize` feature by following here.
```

Hide:

```
RCAdapter.setClosed(true);
// RCAdapter.setClosed(false); // Show
```

Remove:

```
RCAdapter.dispose();
```

Popup the widget

Only for Adapter JS way and popup window feature enabled:

```
RCAdapter.popupWindow();  // popup the widget in a standalone window
```

3.2.2 Login and authorization

Log out user

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-logout'
}, '*');
```

Trigger Login button click

App will open login popup window after getting this command. Follow here to disable popup window, and receive oauth uri.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-login'
}, '*');
```

This command only works when user is not logged in

3.2.3 Navigation

Open the dialer and start a call

Find the widget iframe and use `postMessage` to send command and data:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-new-call',
  phoneNumber: `phone number`,
  toCall: true,
}, '*');
```

This feature can be used for `Click to Dial`. If you set `toCall` to ture, it will start the call immediately.

If you are using Adapter JS way, just you can just call `RCAdapter.clickToCall('phonenumber')`.

Here is tutorial to use RingCentral C2D library to quick implement `Click to Dial` feature.

Navigate to arbitrary path

Navigate to path:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-navigate-to',
  path: '/messages', // '/meeting', '/dialer', '//history', '/settings'
}, '*');
```

Navigate back to previous path:

2.0.0

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-navigate-to',
  path: 'goBack',
}, '*');
```

Go to the SMS tab

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-new-sms',
  phoneNumber: `phone number`,
}, '*');
```

Go to SMS conversation for specific phone number

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-new-sms',
  phoneNumber: `phone number`,
  conversation: true, // will go to conversation page if conversation existed
}, '*');
```

Go to SMS page with prefilled text

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-new-sms',
  phoneNumber: `phone number`,
  text: `your text`
  }]
}, '*');
```

If you are using Adapter JS way, just you can just call `RCAdapter.clickToSMS('phonenumber', 'text')`.

Auto-populate SMS conversation text

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-auto-populate-conversation',
  text: `your text`
}, '*');
```

> **This only works when user is already on the SMS conversation detail page. It will add the specified text into user's conversation input.**

Go to SMS page with prefilled image/attachment

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-new-sms',
  phoneNumber: `phone number`,
  attachments: [{
    name: 'test.txt',
    content: 'data:text/plain;base64,SGVsbG8sIFdvcmxkIQ%3D%3D', // base64 encoded data URI
    }], // optional for sending MMS message with attachments
  }]
}, '*');
```

> **This only works when user is already on the SMS conversation detail page. It will add the specified text into user's conversation input.**

3.2.4 Schedule a meeting

> **Your applicaton needs to have the "Meeting" application scope for this to work.**

```
// meeting info
const meetingBody = {
  topic: "Embbnux Ji's Meeting",
  meetingType: "Scheduled",
  password: "",
  schedule: {
    startTime: 1583312400368,
    durationInMinutes: 60,
    timeZone: {
      id: "1"
    }
  },
  allowJoinBeforeHost: false,
  startHostVideo: false,
  startParticipantsVideo: false,
  audioOptions: [
    "Phone",
    "ComputerAudio"
  ]
};

// send a request to schedule meeting
const requestId = Date.now().toString();
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-message-request',
  requestId: requestId,
  path: '/schedule-meeting',
  body: meetingBody,
}, '*');

// listen response
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-adapter-message-response') {
    if (data.responseId === requestId) {
      console.log(data.response);
    }
  }
});
```

3.2.5 Set presence

1.8.2

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-set-presence',
  userStatus: 'Available', // Offline, Busy, Available
  dndStatus: 'TakeAllCalls', // TakeAllCalls, DoNotAcceptAnyCalls, DoNotAcceptDepartmentCalls, TakeDepartmentCallsOnly
}, '*');
```

To get current presence status please refer this event.

3.2.6 Update ringtone settings

2.0.0

Set ringtone audio

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-update-ringtone',
  name: 'CoolRingTone', // Ringtone name
  uri: 'https://xxx.wav', // Ringtone URI, support http/https and base64 data URI
}, '*');
```

Set ringtone volume

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-update-ringtone',
  volume: 0.5, // 0 - 1.0
}, '*');
```

If you are using Adapter JS way, just you can just call `RCAdapter.updateRingtone({ name, uri, volume })`.

3.3 Alternative authorization methods

RingCentral Embeddable supports the Authorization code with PKCE grant type to facilitate user's logging into RingCentral. This is the recommended authorization method for applications like those built on top of RingCentral Embeddable. Therefore, no changes are necessary to enable authorization and usage of RingCentral Embeddable. However, some developers in specific and rare circumstances may wish to utilize a different method of authorization. This guide will instruct developers on how to do so.

Access tokens are stored in a browser's local storage

In the Authorization code with PKCE flow, a user's access token is managed safely and securely in the browser's local storage for that user. If a user is inactive for more than 7 days, then the user will be automatically logged out. Embeddable automatically refreshes access tokens when API requests are made to the RingCentral API, so as long as the user remains active, they will not be required to login again. Users that are inactive for longer than seven days, however, will be required to login to RingCentral again.

3.3.1 JWT flow

Developers can login to RingCentral Embeddable using the JWT auth flow if they so choose. However, doing so means that every user of RingCentral Embeddable will be logged in as the same user, which may undermine the value of RingCentral's audit trail and security practices. Please use at your own risk.

JWT auth flow in Embeddable is experimental

While the JWT auth flow itself is not experimental, its usage within the context of RingCentral Embeddable is. This is due to the fact that using JWT in this way is beyond the intended design of Embeddable, and could be problematic in some circumstances.

JWT also requires you to expose your client secret, which if exposed publicly could expose you to some security risks.

Javascript iframe

```
<script>
  (function() {
    var rcs = document.createElement("script");
    var clientId = "<YOUR CLIENT ID>";
    var clientSecret = "<YOUR CLIENT SECRET>";
    var jwt = "<YOUR JWT TOKEN>";
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?"+
        "clientId="+clientId+"&clientSecret="+clientSecret+"&jwt="+jwt;
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone"
    src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
clientId=ringcentral_app_client_id&clientSecret=ringcentral_app_client_secret&jwt=your_jwt_token">
</iframe>
```

3.3.2 Authorization code flow

Authorization code flow has been deprecated

RingCentral Embeddable utilizes the Authorization Code with PKCE grant flow by default since `v1.4.0`. Developers are required to upgrade.

If for debugging purposes you need to utilize this mode or authorization, developers can specify their app's client secret using the `clientSecret` URI parameter via a script tag's `src` attribute, or an iframe's `href` attribure.

Pass RingCentral authorization code and code verifier:

Javascript iframe

```
<script>
  (function() {
    var rcs = document.createElement("script");
    var clientId = "<YOUR CLIENT ID>";
    var clientSecret = "<YOUR CLIENT SECRET>";
    var authCode = "<AUTH CODE>";
    var authCodeVerifier = "<AUTH CODE VERIFIER>";
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?"+
       "clientId="+clientId+"&authorizationCode="+authCode+"&authorizationCodeVerifier="+authCodeVerifier;
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>

<iframe width="300" height="500" id="rc-widget" allow="microphone" src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html?
clientId=ringcentral_app_client_id&authorizationCode=ringcentral_authorization_code&authorizationCodeVerifier=code_verifier_for_the_code">
</iframe>
```

PostMessage way

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-authorization-code',
  callbackUri: "http://localhost:8080/redirect.html?"+
     "code=authorization_code&state=MTU5OTE0MzE5NTQ5OQ%3D%3D&code_verifier="
}, '*');
```

`authorizationCodeVerifier` query parameter is only supported after `v1.8.1`

`authorizationCodeVerifier` is not required if you use the authorization URI generated from the login popup event.

For authorization code flow (without PKCE), `clientId` and `clientSecret` is required with `authorizationCode`. The app needs `clientSecret` to exchange token. The authorization code should be generated with same RingCentral app client id and secret.

3.4 Introducing RingCentral Embeddable 2.0

RingCentral Embeddable 2.0, marks the next generation of this popular RingCentral product, allowing developers to easily embed a RingCentral phone, SMS client, and more into any webpage or web application.

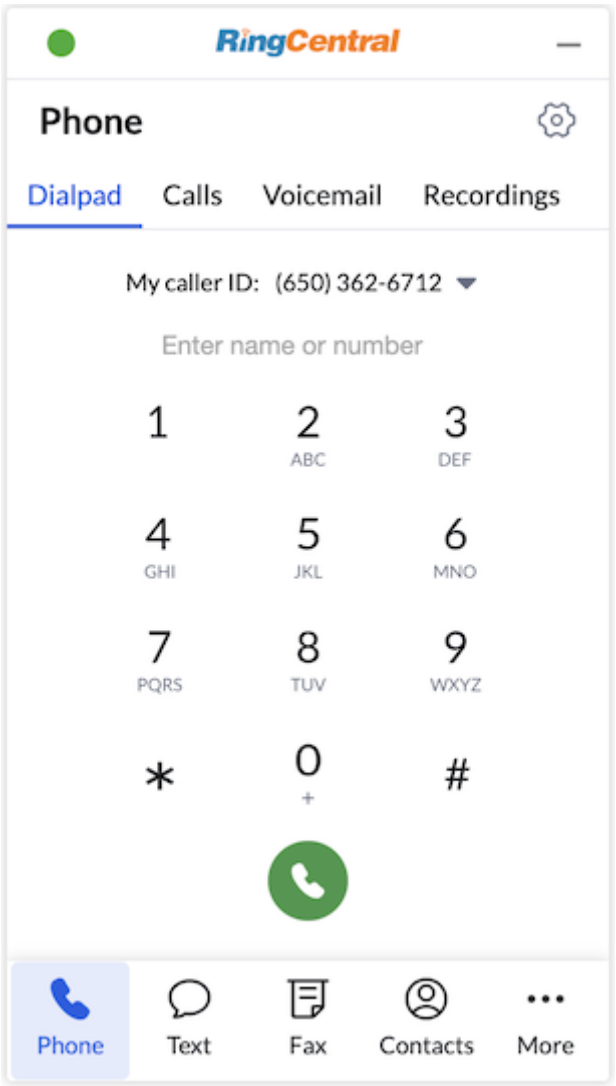
RingCentral Embeddable 2.0 is 100% backwards compatible

The 2.x version of RingCentral Embeddable is completely backwards compatible with all previous versions of Embeddable 1.x. This means that developers can quickly and easily upgrade to the latest version with minimal risk of breaking your existing implementation. PubNub subscription which is deprecated in 1.x is removed in 2.x.

3.4.1 What's new in 2.0

Updated design

RingCentral Embeddable 2.0 front-end has been completely refactored to better conform with RingCentral's user interface guidelines and best practices. Users familiar with the RingCentral



app will feel right at home using the new version of Embeddable. User interface highlights include:

- **New navigation bar**: A redesigned navigation bar for improved accessibility and navigation efficiency.
- **New phone dialer**: The dialer, calls, voicemail and recordings tbas are merged into the phone tab.
- **New call history**: New call history UI provides a more extensible design so developers can add more options for taking action on past calls.
- **New Text inbox**: Experience a refreshed SMS inbox with a modernized user interface, making message management more efficient. Faxes are split into independent tab.
- **New meetings home**: The meetings interface has been updated for a modernized user interface.

Direct access to call recordings

In this new version, a new recordings page has been added, allowing users to conveniently manage and play their recorded calls directly within Embeddable.

To enable this feature, you must have `ReadCallRecording` app scope added in your RingCentral app settings.

SMS templates support

Author responses to common SMS inquiries and share those responses across all employees in your company using RingEX's SMS template feature. This feature require some special setup to fully enable.

Enhanced voicemail player

The new voicemail player in RingCentral Embeddable 2.x comes with seek support, providing users with more control over their voicemail playback.

Noise reduction

Enjoy improved audio quality with noise reduction enabled by default, ensuring a clearer communication experience. This feature is available, but disabled by default in Embeddable 1.x.

Audio settings

RingCentral Embeddable 2.x introduces a new audio settings page, allowing users to easily configure their audio input and output devices. It allows users to configure speaker volume and ringtone device.

Developer features

Over the course of the Embeddable 2.0, we will be working to make the UI more extensible by developers, without developers having to know or code HTML and CSS to conform to the RingCentral UI standard.

Customized settings

RingCentral Embeddable 2.x introduces a new customized settings API, offering developers more flexibility to tailor Embeddable's settings to the specific needs of the application it powers. Consider for example a circumstance in which Embeddable is being used within a third-party app, and settings specific to that app need to be made available to end users. This feature allows developers to inject the setting/preference into the "Settings" page using a simple JSON data structure.

Customized pages and tabs

RingCentral Embeddable 2.x allows developers to register custom tabs and custom pages into the widget using a simple JSON schema. This allows developers to extend the user interface of Embeddable in countless ways. Generate fully customized forms to prompt users for input, or create custom listing screens to allow users to scroll and search data sources.

3.4.2 Try it

If you are using latest build, you will be upgraded to 2.x version automatically. If you want to use fixed version uri, you can upgrade manually:

Javascript **iframe**

Add following code to any website's header to embed a RingCentral phone into that page.

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/2.2.0/adapter.js?clientId=YOUR_RINGCENTRAL_CLIENT_ID";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

Add the following anywhere on your webpage.

```
<iframe width="300" height="500" allow="microphone"
  src="https://apps.ringcentral.com/integration/ringcentral-embeddable/2.2.0/app.html?clientId=YOUR_RINGCENTRAL_CLIENT_ID">
</iframe>
```

Then add following redirect URI into your app settings in RingCentral Developer Console:

```
https://apps.ringcentral.com/integration/ringcentral-embeddable/2.2.0/redirect.html
```

3.5 Service features

3.5.1 Service integration features

After integrating the RingCentral Embeddable library into your web application, you can also integrate your own custom service into the CTI as well. This will allow you to associate an icon with contacts you synchronize into Embeddable via its API, or display a button to facilitate authorization with your service. In a nutshell, anywhere in Embeddable where the library allows you to modify or augment the user interface, requires you to first register your service so the Embeddable can properly indicate what features are native to the CTI, and which ones have been added by a third-party.

Registering your app as a service in Embeddable

The code below shows how to register your service. When you do so you will choose a name for your service, below we use `TestService`. You will use that exact same name when engaging with the service API features. You will register your service by using the `postMessage` API.

Registering your service via postMessage

```
var registered = false;
window.addEventListener('message', function (e) {
  const data = e.data;
  // Register when widget is loaded
  if (data && data.type === 'rc-login-status-notify' && registered === false) {
    registered = true;
    document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
      type: 'rc-adapter-register-third-party-service',
      service: {
        name: 'TestService'
      }
    }, '*');
  }
});
```

3.5.2 Integration contacts in your system into Embeddable

This feature requires you to register your app as a service first.

Show contacts on Embeddable's contacts tab

First you need to pass `contactsPath` when you register service:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    contactsPath: '/contacts',
    contactIcon: 'https://set_brand_icon.com/test.png', // optional, show brand icon in the top right of contact avatar
  }
}, '*');
```

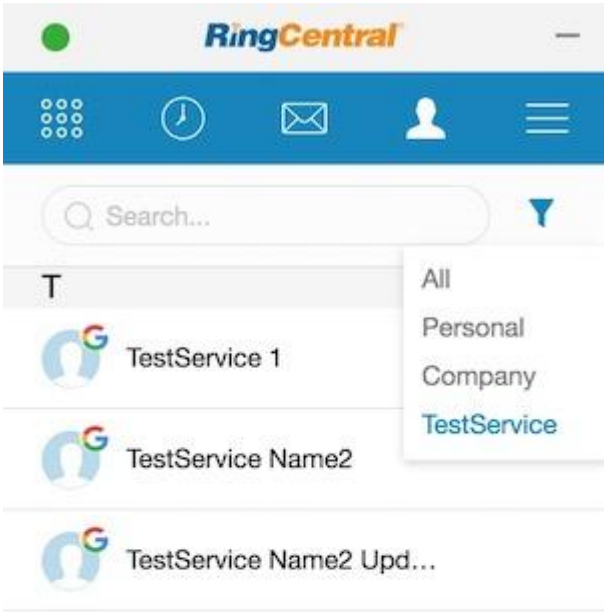
Add a message event to respond to a contacts list event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/contacts') {
      console.log(data);
      // you can get page and syncTimestamp params from data.body
      // query contacts data from third party service with page and syncTimestamp
      // if syncTimestamp existed, please only return updated contacts after syncTimestamp
      // response to widget:
      const contacts = [{
        id: '123456', // id to identify third party contact, need to be string
        name: 'TestService Name', // contact name
        type: 'TestService', // need to same as service name
        phoneNumbers: [{
          phoneNumber: '+1234567890',
          phoneType: 'direct', // support: business, extension, home, mobile, phone, unknown, company, direct, fax, other
        }],
        company: 'CompanyName',
        jobTitle: 'Engineer',
        emails: ['test@email.com'],
        profileImageUrl: 'https://avatar_uri', // optional, show avatar in Contacts page
        deleted: false, // set deleted to true if you need to delete it in updated contacts
      }];
      // pass nextPage number when there are more than one page data, widget will repeat same request with nextPage increased
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: {
          data: contacts,
          nextPage: null,
          syncTimestamp: Date.now()
        },
      }, '*');
    }
  }
});
```

Data from `contactsPath` will be displayed in the contacts tab inside Embeddable.

Embeddable will request contacts data when widget is loaded and when user visit contacts page. In first request `syncTimestamp` is blank, so you need to provide full contacts data to widget. Please provide `syncTimestamp` when reponse to widget. In next contacts request widget will send you `syncTimestamp`, so you just need to provide updated contact since `syncTimestamp`.

If you provide `nextPage` for `contactsPath` response, widget will repeat request with `page="${nextPage}"` to get next page contacts data.



To trigger contact sync request in widget using the API:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-sync-third-party-contacts',
}, '*');
```

Show contacts search result in the dialer's receiver input

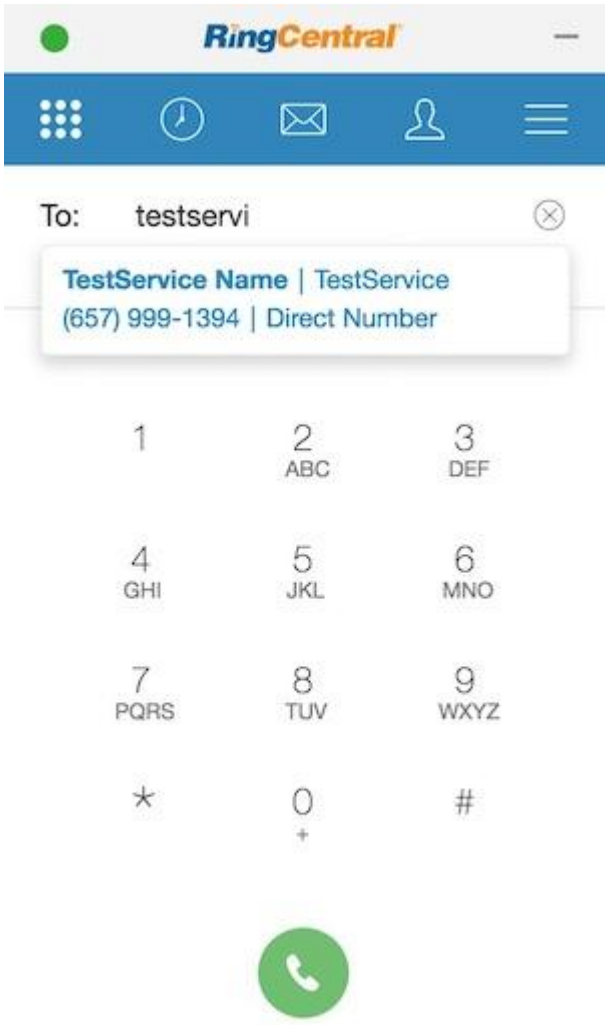
You must want to show related contacts result when user typing in callee input area. First you need to pass `contactSearchPath` when you register service:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    contactSearchPath: '/contacts/search'
  }
}, '*');
```

Add a message event to response contacts search event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/contacts/search') {
      console.log(data);
      const searchedContacts = [{
        id: '123456', // id to identify third party contact
        name: 'TestService Name',
        type: 'TestService', // need to same as service name
        phoneNumbers: [{
          phoneNumber: '+1234567890',
          phoneType: 'direct', // support: business, extension, home, mobile, phone, unknown, company, direct, fax, other
        }]
```

```
      }];
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: {
          data: searchedContacts
        },
      }, '*');
    }
  }
});
```



Show matched contact in the call history or incoming call page

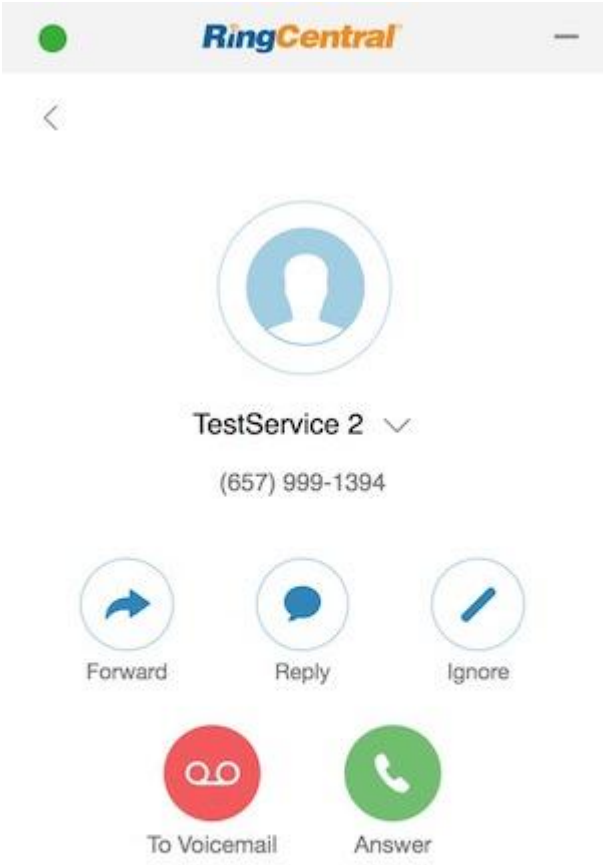
In widget, we use contact matcher to match phone number to contact in calls page or incoming page. First you need to pass `contactMatchPath` when you register service:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    contactMatchPath: '/contacts/match',
    contactMatchTtl: 2 * 60 * 60 * 1000, // optional, contact match data cache deleted time in seconds, default is 2 hours, supported from v1.10.2
    contactNoMatchTtl: 5 * 60 * 1000, // optional, contact match data expired in seconds, will re-match at next match trigger, default is 5 minutes, from
v1.10.2
  }
}, '*');
```

Add a message event to response contacts matcher event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
```

```
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/contacts/match') {
      console.log(data); // include phone number array that need to match
      const matchedContacts = {
        '+12165325078': [
          {
            id: '123456', // id to identify third party contact
            type: 'TestService', // need to same as service name
            name: 'TestService 1',
            phoneNumbers: [{
              phoneNumber: '+12165325078',
              phoneType: 'direct', // support: business, extension, home, mobile, phone, unknown, company, direct, fax, other
            }]
          }
        ]
      };
      // return matched contact object with phone number as key
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: {
          data: matchedContacts
        },
      }, '*');
    }
  }
});
```



TRIGGER CONTACT MATCH MANUALLY

2.0.0

If there are new contacts in your system, you can trigger contact match manually:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-trigger-contact-match',
  phoneNumbers: [`phoneNumberInE164Format`],
}, '*');
```

VIEW MATCHED CONTACT EXTERNALLY

2.0.0

You can also view matched contact in your system by clicking "View contact details" in the call history or inbox page. You need to pass `viewMatchedContactPath` when you register service:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    // ...
    contactMatchPath: '/contacts/match',
    viewMatchedContactPath: '/contacts/view',
  }
}, '*');
```

Add a message event to response view matched contact event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    // ... match contact event
    // ...
    if (data.path === '/contacts/view') {
      console.log(data.body); // contact info to view
      // open contact detail page in your system
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: 'ok',
      }, '*');
    }
  }
});
```

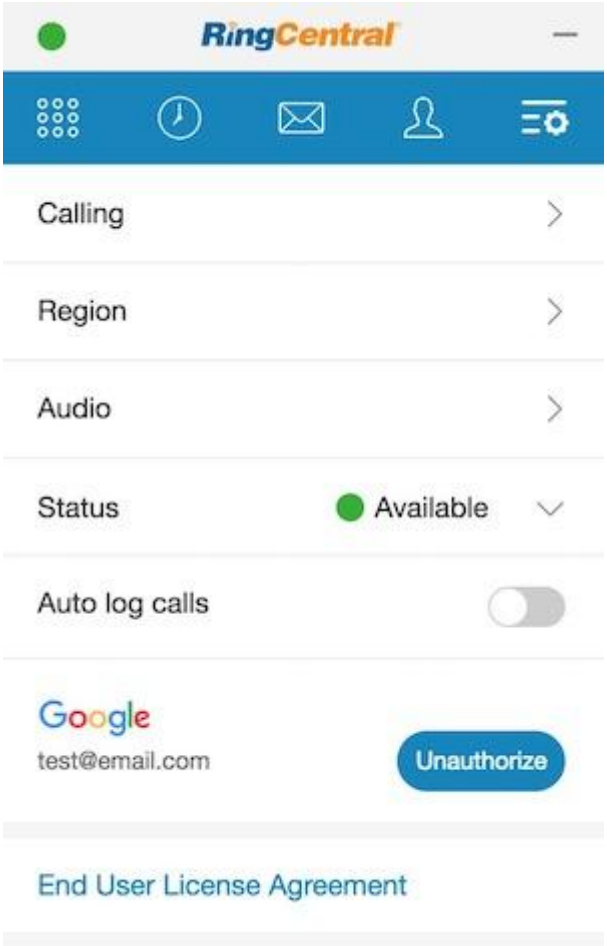
3.5.3 Add third-party authorization button

For some CRM API, they requires user to authorize firstly. This feature allows developer to add a third party authorization button and sync status into widget.

First you need to pass `authorizationPath`, `authorizedTitle`, `unauthorizedTitle` and `authorized` when you register service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    displayName: 'TestServiceDisplayName', // Optional, supported from 2.0.1
    info: 'Some description about this service', // Optional, supported from 2.0.0
    authorizationPath: '/authorize',
    authorizedTitle: 'Unauthorize',
    unauthorizedTitle: 'Authorize',
    authorized: false,
    authorizedAccount: 'test@email.com', // optional, authorized account email or id
    authorizationLogo: 'https://your_brand_picture/logo.png', // optional, show your brand logo in authorization section, recommended: height 30px, width <
85px.
    // showAuthRedDot: true, // optional, this will show red dot at settings page when need to auth
  }
}, '*');
```

After registered, you can get a `TestService authorization button` in setting page:



Add a message event to response authorization button event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/authorize') {
      // add your codes here to handle third party authorization
      console.log(data);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
```

```
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

Update authorization status in widget:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-update-authorization-status',
  authorized: true,
  authorizedAccount: 'test@email.com', // optional, authorized account email or id
}, '*');
```

If you register an authorization service into Embeddable, the contacts-related service above will work only after the user's status has changed to authorized.

3.5.4 Show contact's activities from your application

> **This feature requires you to register your app as a service first.**

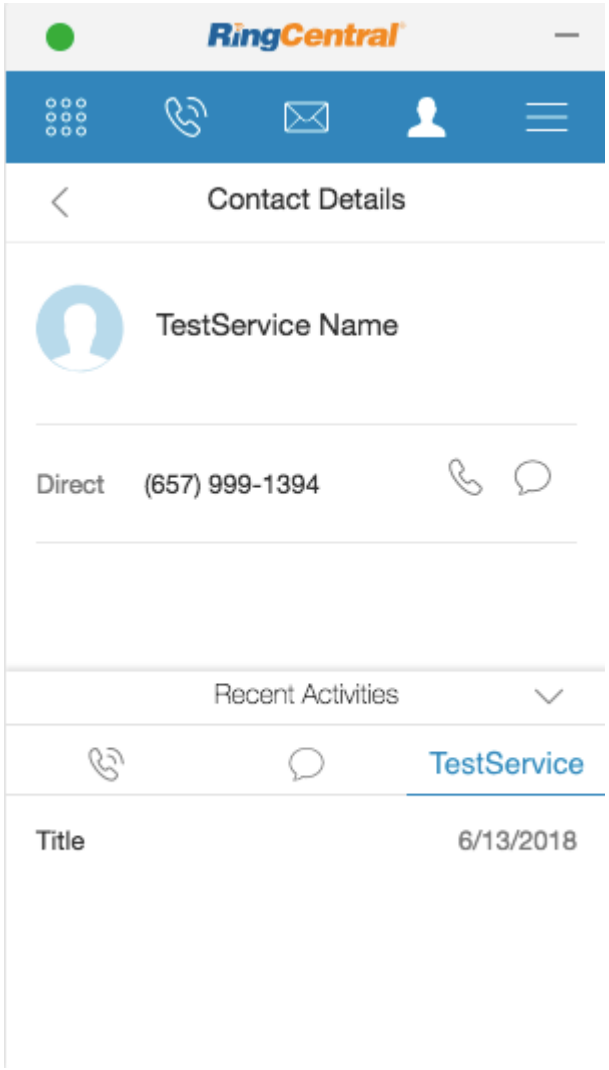
First you need to pass `activitiesPath` and `activityPath` when you register service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    activityName: 'TestService', // optional, will use service.name as default
    activitiesPath: '/activities',
    activityPath: '/activity'
  }
}, '*');
```

Add a message event to response activities query event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/activities') {
      const contact = data;
      console.log(contact);
      const activities = [{
        id: '123',
        subject: 'Title',
        time: 1528854702472
      }];
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: activities },
      }, '*');
    }
    if (data.path === '/activity') {
      const activity = data;
      // handle activity here
      console.log(activity);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

Data from `activitiesPath` will be showed in contact details page in the widget. Event from `activityPath` is triggered when user click activity item in the widget.



3.5.5 Log a call in your service

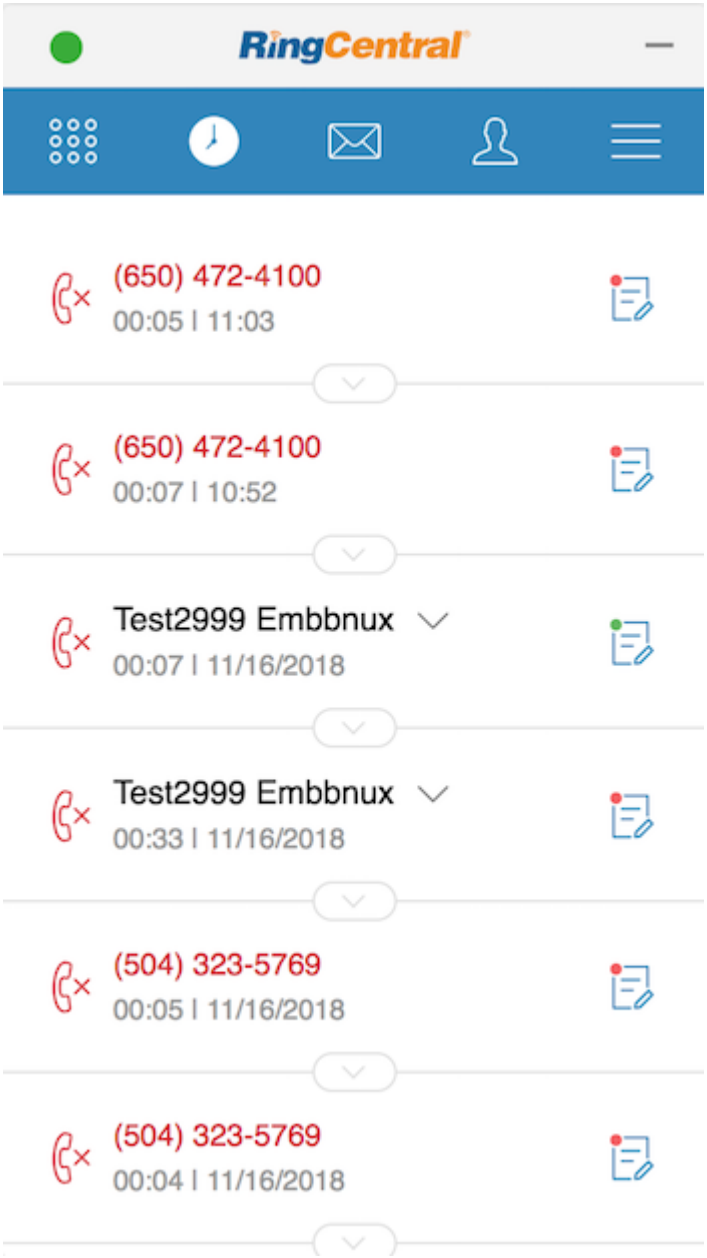
> **This feature requires you to register your app as a service first.**

Add call log button/icon in call history tab

First you need to pass `callLoggerPath` and `callLoggerTitle` when you register service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    callLoggerPath: '/callLogger',
    callLoggerTitle: 'Log to TestService',
    // callLoggerAutoSettingLabel: 'Auto log calls', // optional, customized the auto log setting label
    // recordingWithToken: 1
  }
}, '*');
```

After registered, you can get a `Log to TestService` in calls page, and `Auto log calls` setting in setting page



Then add a message event to response call logger button event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/callLogger') {
      // add your codes here to log call to your service
      console.log(data);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

This message event is fired when user clicks `Log` button. Or if user enables `Auto log calls` in settings, this event will be also fired when a call is started and updated.

In this message event, you can get call information in `data.body.call`. When call is recorded and recording file is generated, you can get `recording` data in `data.body.call`:

```
{
  contentUri: "https://media.devtest.ringcentral.com/restapi/v1.0/account/170848004/recording/6469338004/content"
  id: "6469338004"
  link: "http://apps.ringcentral.com/integrations/recording/sandbox/?id=Ab7937-59r6EzUA&recordingId=6469338004"
  type: "OnDemand"
  uri: "https://platform.devtest.ringcentral.com/restapi/v1.0/account/170848004/recording/6469338004"
}
```

The `link` property in `recording` is a link to get and play recording file from RingCentral server. The `contentUri` is a URI which can be used to get `recording` file with RingCentral access token. If you pass `recordingWithToken` when register service, you can get contentUri with `access_token`. The `access_token` will be expired in minutes, so need to download immediately when get it.

```
{
  contentUri: "https://media.devtest.ringcentral.com/restapi/v1.0/account/170848004/recording/6469338004/content?access_token=ringcentral_access_token"
  id: "6469338004"
  link: "http://apps.ringcentral.com/integrations/recording/sandbox/?id=Ab7937-59r6EzUA&recordingId=6469338004"
  type: "OnDemand"
  uri: "https://platform.devtest.ringcentral.com/restapi/v1.0/account/170848004/recording/6469338004"
}
```

Auto log calls setting

1.10.0

User can enable/disable auto log in settings page. To set default `Auto log calls` enabled:

Add `defaultAutoLogCallEnabled` into the `adapter.js` URI:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?defaultAutoLogCallEnabled=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

Listen to `Auto log calls` setting changed:

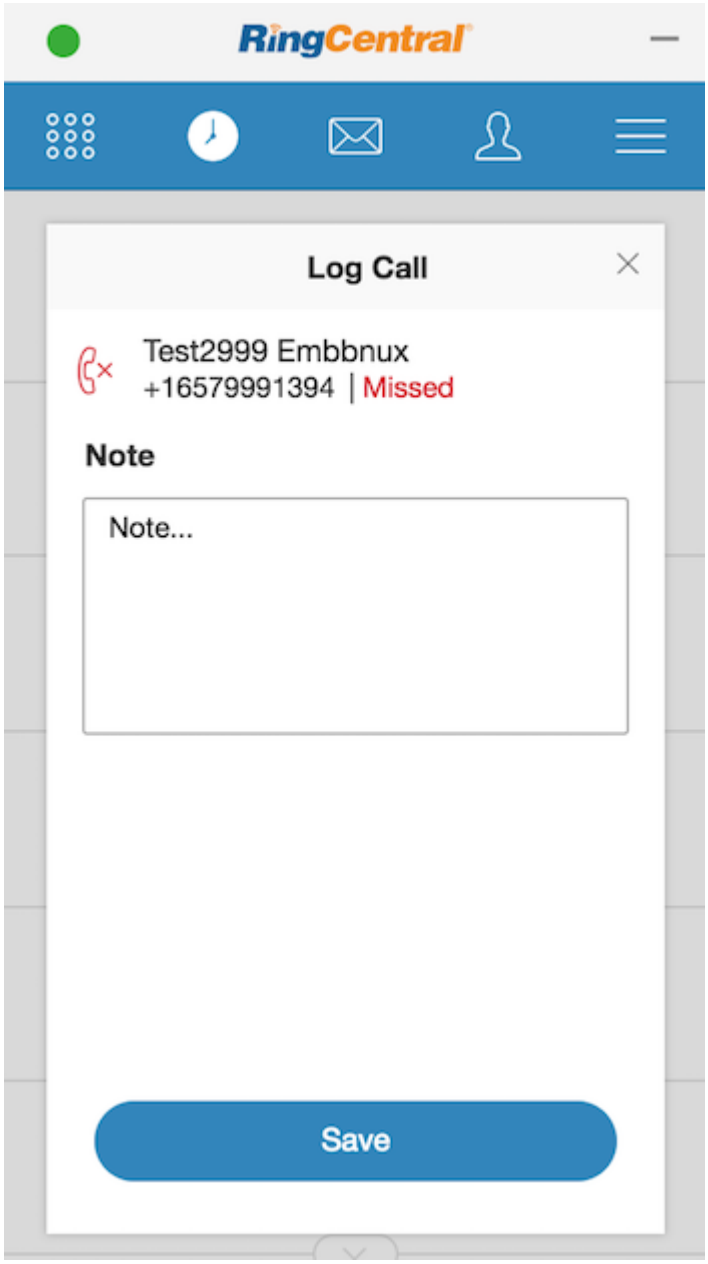
```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-callLogger-auto-log-notify') {
    console.log('rc-callLogger-auto-log-notify:', data.autoLog);
  }
});
```

Add call logger modal

For some developers who want to add note when log a call to their platform, we provide a log modal to support it.

Add `showLogModal` when register service:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    callLoggerPath: '/callLogger',
    callLoggerTitle: 'Log to TestService',
    showLogModal: true,
  }
}, '*');
```



Add call log entity matcher

In call logger button, widget needs to know if call is logged. To provide `callLogEntityMatcherPath` when register, widget will send match request to get match result of calls history.

Note: If you have third party auth configured, call log entity matcher only works when `authorized` is `true`.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    callLoggerPath: '/callLogger',
    callLoggerTitle: 'Log to TestService',
    callLogEntityMatcherPath: '/callLogger/match'
  }
}, '*');
```

Then add a message event to response call logger matcher event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
```

```
    if (data.path === '/callLogger/match') {
      // add your codes here to reponse match result
      console.log(data); // get call session id list in here
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: {
          data: {
            '214705503020': [{ // call session id from request
              id: '88888', // call log entity id from your platform
              note: 'Note', // Note of this call log entity
            }]
          }
        },
      }, '*');
    }
  }
});
```

TRIGGER CALL LOGGER ENTITY MATCH MANUALLY

The widget will trigger call logger entity match after call logged automatically. But you can still trigger it to match manually

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-trigger-call-logger-match',
  sessionIds: [`call_session_id`],
}, '*');
```

Get un-logged calls

2.1.0

When user have calls in other device during the widget closed, those calls data can't be sent by the `callLoggerPath` event even auto log enabled. You can get those calls by un-logged calls api.

```
const { calls, hasMore } = await RCAdapter.getUnloggedCalls(PER_PAGE, PAGE_NUMBER); // PER_PAGE: number of calls per page, PAGE_NUMBER: page number
```

3.5.6 Log RingCentral video meeting into your service

This feature requires you to register your app as a service first.

This is only relevant for customers who use RingCentral Video

First you need to pass `meetingLoggerPath` and `meetingLoggerTitle` when you register service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    meetingLoggerPath: '/meetingLogger',
    meetingLoggerTitle: 'Log to TestService',
  }
}, '*');
```

After registered, you can get a `Log to TestService` in meeting history page.

Then add a message event to response meeting logger button event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/meetingLogger') {
      // add your codes here to log meeting to your service
      console.log(data);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

3.5.7 Log messages into your service

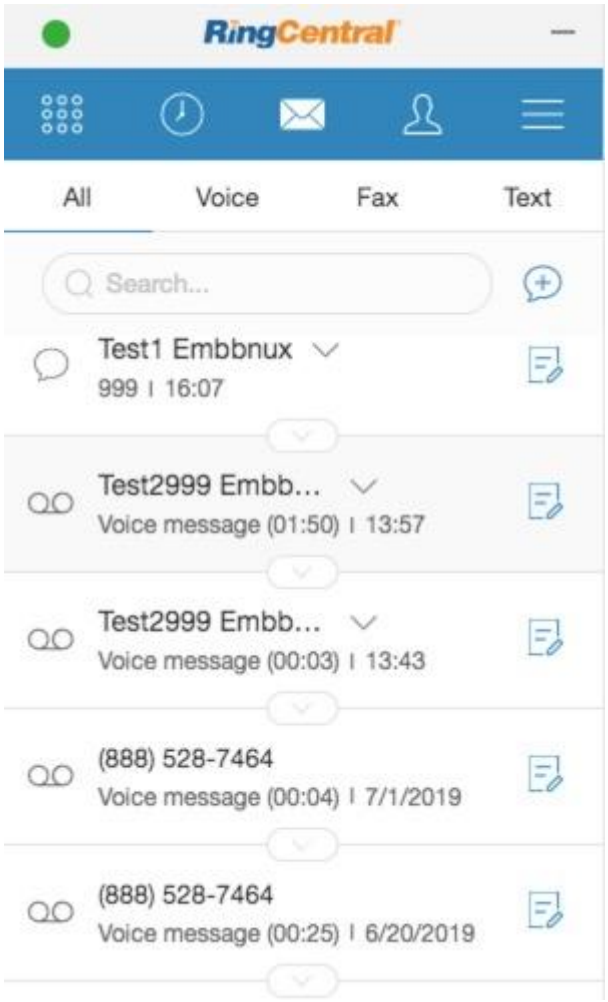
> **This feature requires you to register your app as a service first.**

Add message logger button in messages page

First you need to pass `messageLoggerPath` and `messageLoggerTitle` when you register service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    messageLoggerPath: '/messageLogger',
    messageLoggerTitle: 'Log to TestService',
    // messageLoggerAutoSettingLabel: 'Auto log messages', // optional, customize the auto log setting label
    // attachmentWithToken: true,
  }
}, '*');
```

After registered, you can get a `Log to TestService` in messages page, and `Auto log messages` setting in setting page:



Then add a message event to response message logger button event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/messageLogger') {
      // add your codes here to log messages to your service
      console.log(data);
      // response to widget
```

```
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

This message event is fired when user clicks `Log` button. Or if user enables `Auto log messages` in settings, this event will be also fired when a message is created and updated.

In this message event, you can get call information in `data.body.conversation`. Messages are grouped by `conversationId` and `date`. So for a conversation that have messages in different date, you will receive multiple log message event.

For Voicemail and Fax, you can get `attachment` data in message. The `attachment.link` is a link used to get voicemail file from RingCentral server with Browser. The `attachment.uri` is a URI which can be used to get attachment file with RingCentral access token. If you pass `attachmentWithToken` when register service, you can get `attachment.uri` with `access_token`. The `access_token` will be expired in minutes, so need to download immediately when get it.

Auto log messages settings

1.10.0

User can enable/disable auto log in settings page. To set default `Auto log messages` enabled:

Add `defaultAutoLogMessageEnabled` into the `adapter.js` URI:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js?defaultAutoLogMessageEnabled=1";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

Listen to `Auto log messages` setting changed:

2.0.0

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-messageLogger-auto-log-notify') {
    console.log('rc-messageLogger-auto-log-notify:', data.autoLog);
  }
});
```

Add message log entity matcher

In message logger, widget needs to know if messages are logged. To provide `messageLogEntityMatcherPath` when register, widget will send match request to get match result of messages history.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    messageLoggerPath: '/callLogger',
    messageLoggerTitle: 'Log to TestService',
    messageLogEntityMatcherPath: '/messageLogger/match'
  }
}, '*');
```

Then add a message event to response message logger match event:

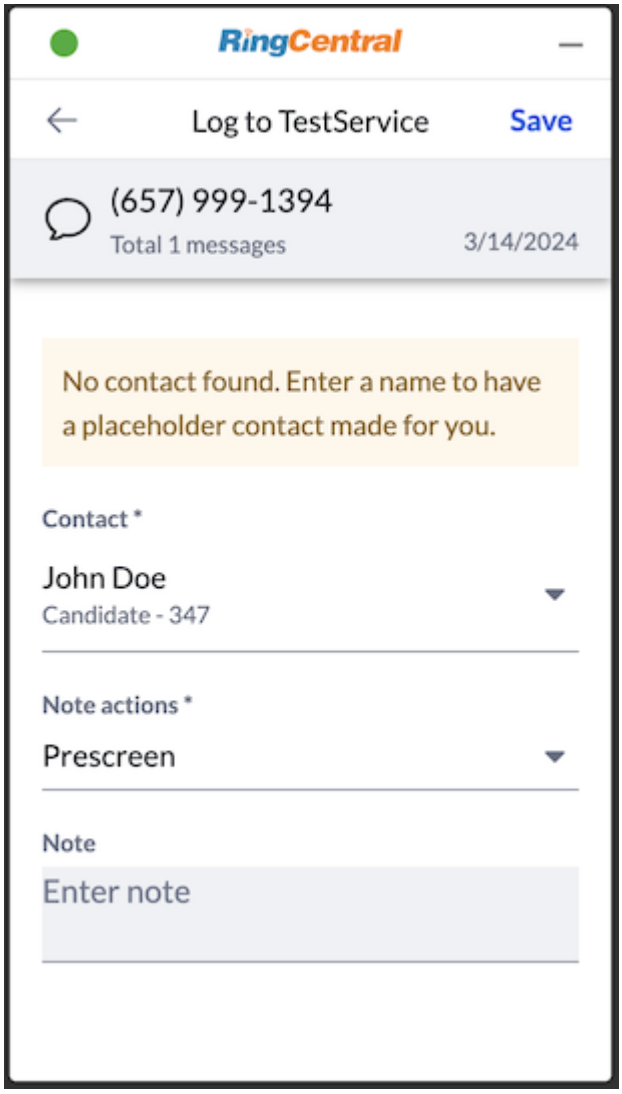
```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/messageLogger/match') {
      // add your codes here to reponse match result
      console.log(data); // get message conversation log id list in here
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
```

```
        responseId: data.requestId,
        response: {
          data: {
            '6740354775690179O5/7/2/2019': [{ // conversation log id from request
              id: '88888', // log entity id from your platform
            }]
          }
        },
      }, '*');
    }
  }
});
```

Message log page

2.0.0

You can also add a message log page to have an related form when user logs messages to your service.



Register message log service:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    messageLoggerPath: '/messageLogger',
    messageLoggerTitle: 'Log to TestService',
    messagesLogPageInputChangedEventPath: '/messageLogger/inputChanged',
  }
}, '*');
```

Then add message event listener to show message log page and input changed request:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/messageLogger') {
      // Get trigger type: data.body.triggerType
      // When user click log button in message item, triggerType is 'manual'
      // When user enable auto log, triggerType is 'auto' for new message
      if (data.body.triggerType === 'manual') {
        // customize message log page
        document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
          type: 'rc-adapter-update-messages-log-page',
          page: {
            title: 'Log to TestService',
            // schema and uiSchema are used to customize call log page, api is the same as [react-jsonschema-form](https://rjsf-team.github.io/react-
jsonschema-form)
            schema: {
              type: 'object',
              required: ['contact', 'noteActions'],
              properties: {
                "warning": {
                  "type": "string",
                  "description": "No contact found. Enter a name to have a placeholder contact made for you.",
                },
                "contact": {
                  "title": "Contact",
                  "type": "string",
                  "oneOf": [
                    {
                      "const": "xxx",
                      "title": "John Doe",
                      "description": "Candidate - 347",
                    },
                    {
                      "const": "newEntity",
                      "title": "Create placeholder contact"
                    }
                  ],
                },
                "contactName": {
                  "type": 'string',
                  "title": "Contact name",
                },
                "contactType": {
                  "title": "Contact type",
                  "type": "string",
                  "oneOf": [
                    {
                      "const": "candidate",
                      "title": "Candidate"
                    },
                    {
                      "const": "contact",
                      "title": "Contact"
                    }
                  ],
                },
                "noteActions": {
                  "type": "string",
                  "title": "Note actions",
                  "oneOf": [
                    {
                      "const": "prescreen",
                      "title": "Prescreen"
                    },
                    {
                      "const": "interview",
                      "title": "Interview"
                    }
                  ],
                },
                "note": {
                  "type": "string",
                  "title": "Note"
                },
              }
            },
            uiSchema: {
              warning: {
                "ui:field": "admonition", // or typography to show raw text
                "ui:severity": "warning", // "warning", "info", "error", "success"
              },
              contactName: {
                "ui:placeholder": 'Enter name',
                "ui:widget": "hidden", // remove this line to show contactName input
              },
              contactType: {
                "ui:placeholder": 'Select contact type',
                "ui:widget": "hidden", // remove this line to show contactName input
              },
              note: {
                "ui:placeholder": 'Enter note',
```

```
              "ui:widget": "textarea",  // show note input as textarea
            },
            submitButtonOptions: {
              submitText: 'Save',
            },
          },
          formData: {
            contact: 'xxx',
            contactName: '',
            contactType: '',
            noteActions: 'prescreen',
            note: '',
          },
        }
      }, '*');
      // navigate to message log page
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-adapter-navigate-to',
        path: `/log/messages/${data.body.conversation.conversationId}`, // conversation id that you received from message logger event
      }, '*');
    }
    if (data.body.triggerType === 'logForm' || data.body.triggerType === 'auto') {
      // Save message log to your platform
      console.log(data.body); // data.body.conversation, data.body.formData
    }
    // response to widget
    document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
      type: 'rc-post-message-response',
      responseId: data.requestId,
      response: { data: 'ok' },
    }, '*');
  }
  if (data.path === '/messageLogger/inputChanged') {
    console.log(data); // get input changed data in here: data.body.formData
    document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
      type: 'rc-post-message-response',
      responseId: data.requestId,
      response: { data: 'ok' },
    }, '*');
    // you can update message log page data here to make the form dynamic
    return;
  }
 }
});
```

3.5.8 Add meeting schedule feature with your service

This feature requires you to register your app as a service first.

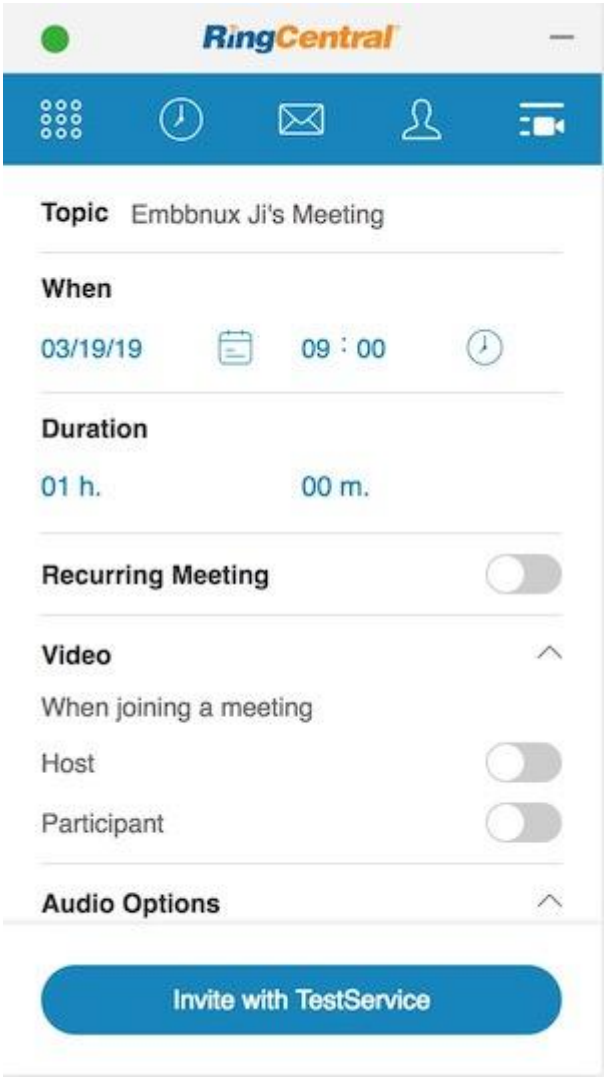
Deprecated. This is only relevant for customers who use RingCentral Meetings

First we need to add `Meeting` permission into your app in RingCentral Developer website if you are using your own RingCentral client id. This works on RingCentral Video or RingCentral Meetings service.

Then pass `meetingInvitePath` and `meetingInviteTitle` when you register service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService', // service name
    meetingInvitePath: '/meeting/invite',
    meetingInviteTitle: 'Invite with TestService',
  }
}, '*');
```

After registered, we can get `Schedule Meeting` page in navigator, and `Invite` button in meeting page:



Add a message event to response meeting invite button event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/meeting/invite') {
      // add your codes here to handle meeting invite data
      console.log(data);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

3.5.9 Show upcoming meeting list in RingCentral Video page

This feature requires you to register your app as a service first.

This only works on RingCentral Video meeting service.

First you need to pass `meetingUpcomingPath` when you register meeting invite service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService', // service name
    meetingInvitePath: '/meeting/invite',
    meetingInviteTitle: 'Invite with TestService',
    meetingUpcomingPath: '/meetingUpcomingList
  }
}, '*');
```

Then add a message event to response upcoming meeting request:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/meetingUpcomingList') {
      // add your codes here to query upcoming meeting from your service
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: {
          data: [{
            id: '123456',
            title: 'Test Meeting in TestService',
            editEventUrl: 'https://your_meeting_uri',
            startTime: "2020-03-22T01:00:00Z",
            endTime: "2020-03-22T02:00:00Z",
            meetingIds: ['433214948'],
          }],
        },
      }, '*');
    }
  }
});
```

3.5.10 VCard click handler

> **This feature requires you to register your app as a service first.**

In SMS messages, user can receive vcard (contact) file with MMS. We allow third party to handle the vard attachment download event. For example, when user click vcard file download button, your service will receive the vcard URI, and save the contact into your service.

First, register service with `vcardHandlerPath`:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    vcardHandlerPath: '/vcardHandler',
  }
}, '*');
```

Add a message event to listen vcard click event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/vcardHandler') {
      // add your codes here to handle the vcard file download event
      console.log(data.body.vcardUri);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

3.5.11 SMS toolbar button

First, register service with `buttonEventPath`:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    buttonEventPath: '/button-click',
    buttons: [{
      id: 'template',
      type: 'smsToolbar',
      icon: 'icon_url',
      label: 'Template',
    }],
  }
}, '*');
```

Add a message event to listen button click event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/button-click') {
      // add your codes here to handle the vcard file download event
      console.log(data.body.button);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

3.5.12 Do Not Contact

2.0.0

This is feature to prevent the user to call/message someone who is in the DoNotContact list in your service.

To enable the feature, please pass `doNotContactPath` when you register service.

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    doNotContactPath: '/doNotContact',
  }
}, '*');
```

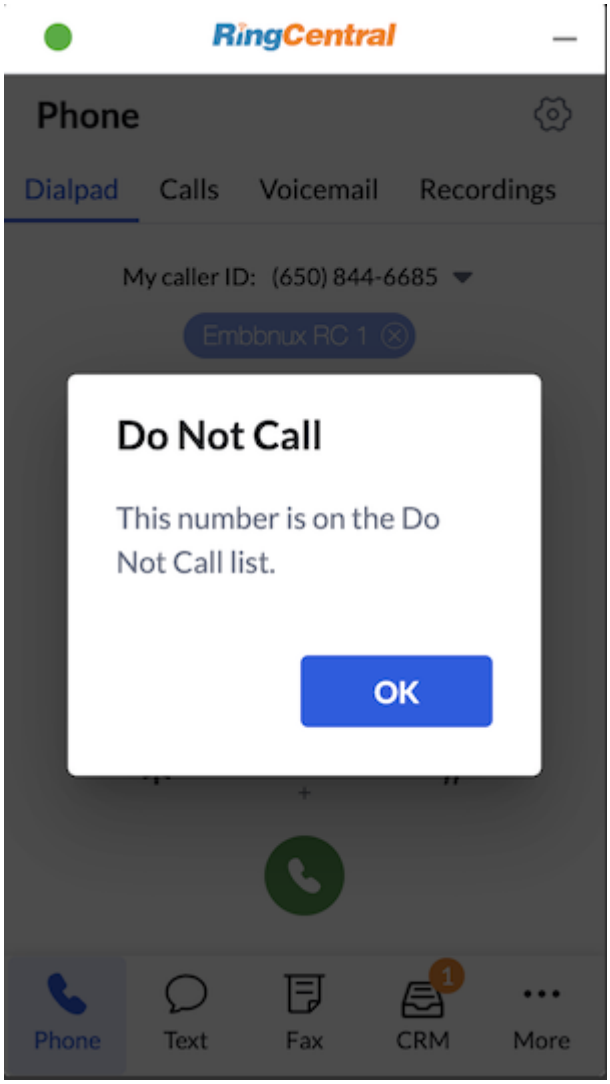
Add a message event to response DoNotContact checking event:

```
// Function to response message to widget
function responseMessage(request, response) {
  console.log(request);
  document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
    type: 'rc-post-message-response',
    responseId: request.requestId,
    response,
  }, '*');
}

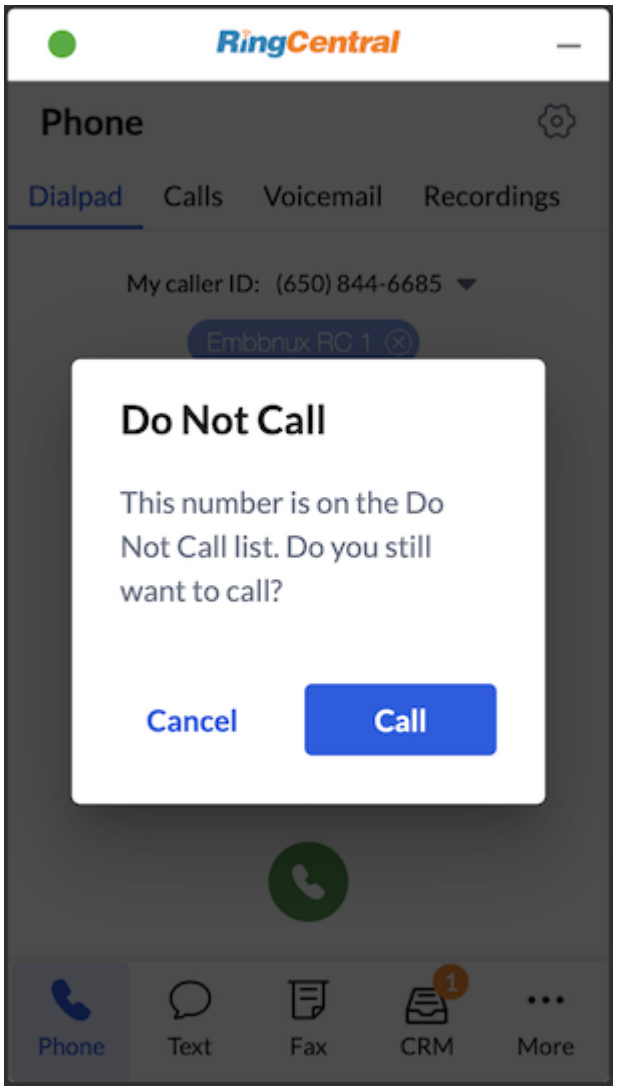
const DO_NOT_CONTACTS_LIST = []; // Your DoNotContact list
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    const request = data;
    if (request.path === '/doNotContact') {
      // For DoNotContact checking for call
      if (request.body.actionType === 'call') {
        // Check if the phone number is in the DoNotContact list, you can check with your own API/logic
        if (DO_NOT_CONTACTS_LIST.includes(request.body.phoneNumber)) {
          responseMessage(request, {
            data: {
              result: true, // true: do not contact, false: can contact,
              message: 'This is a do not contact message.', // optional, message to show in widget
              mode: 'restrict', // optional, restrict mode to prevent user from calling. Or allow user to force call after warning.
            },
          });
          return;
        }
        // If the phone number is not in the DoNotContact list, you can allow the user to call
        responseMessage(request, {
          data: {
            result: false,
          },
        });
        return;
      }
    }
    // For DoNotContact checking for sms
    if (request.body.actionType === 'sms') {
      // Check if the phone number is in the DoNotContact list, you can check with your own API/logic
      if (request.body.recipients.find((item) => DO_NOT_CONTACTS_LIST.includes(item.phoneNumber))) {
        responseMessage(request, {
          data: {
            result: true, // true: do not contact, false: can contact,
            message: 'This is a do not contact message',
            mode: 'restrict' // optional, restrict mode to prevent user from messaging. Or allow user to force sending after warning.
          },
        });
        return;
      }
      responseMessage(request, {
        data: {
          result: false,
        },
      });
      return;
    }
  }
});
```

When user make a call or send a message, the widget will send request to check if the phone number is in the DoNotContact list. If the phone number is in the DoNotContact list, the widget will show a warning message to the user and prevent the user from calling/messaging. If the phone number is not in the DoNotContact list, the widget will allow the user to call/message.

Restrict mode:



No restrict mode:

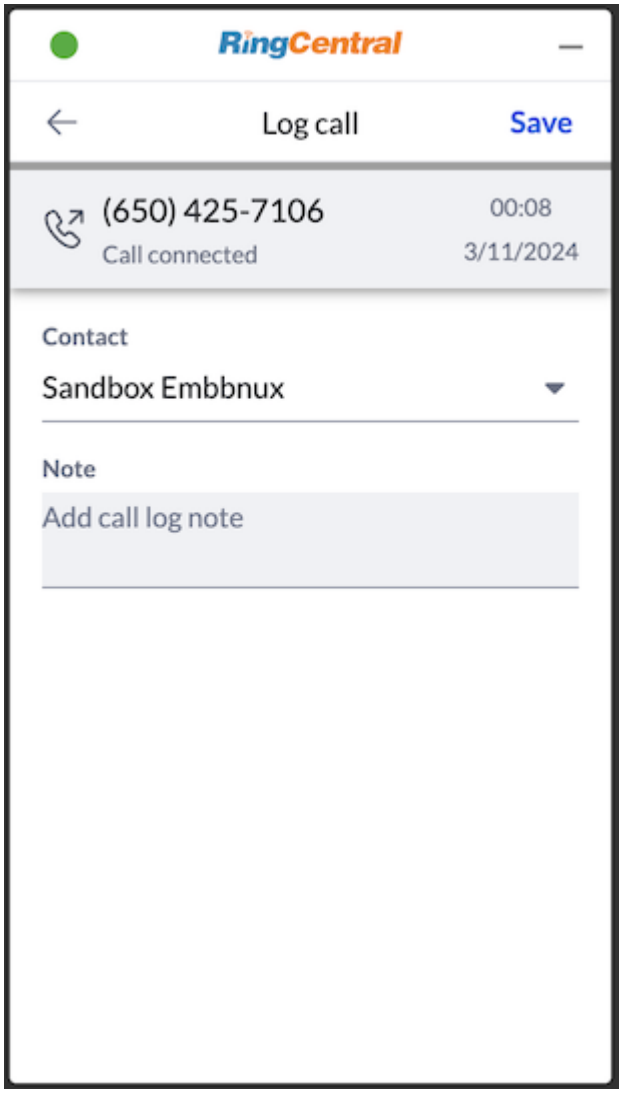


3.6 User interface customization

3.6.1 Creating a customize call log page

2.0.0

From `v2.0.0`, call logger modal is refactored into call log page:



You can customize call log page by adding `callLogPageDataPath` and `callLogPageInputChangedEventPath` when register service:

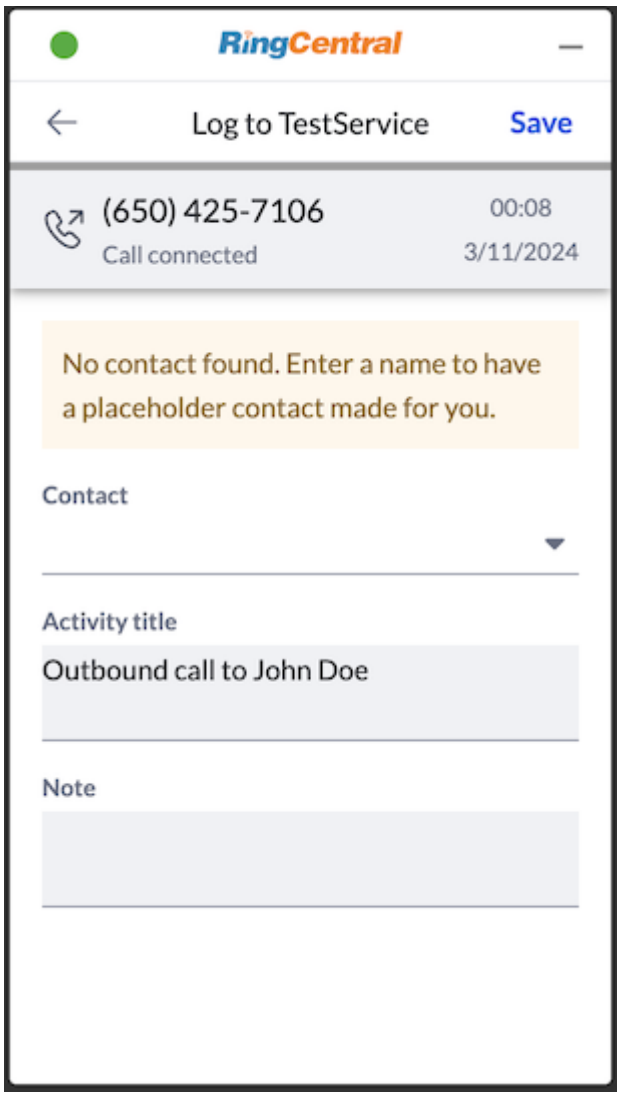
```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    callLoggerPath: '/callLogger',
    callLoggerTitle: 'Log to TestService',
    // showLogModal: false, // disable showLogModal if you want to use call log page
    callLogPageInputChangedEventPath: '/callLogger/inputChanged',
  }
}, '*');
```

Then add message event listener to show call log page and input changed request:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/callLogger') {
```

```
      // Get trigger type: data.body.triggerType
      // When user click log button in call item, triggerType is 'createLog' or 'editLog'
      // When it is triggered from auto log, triggerType is 'presenceUpdate'
      // When save button clicked, triggerType is 'logForm'
      if (data.body.triggerType === 'createLog' || data.body.triggerType === 'editLog') {
        // customize call log page
        document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
          type: 'rc-adapter-update-call-log-page',
          page: {
            title: 'Log to TestService',
            // schema and uiSchema are used to customize call log page, api is the same as [react-jsonschema-form](https://rjsf-team.github.io/react-
jsonschema-form)
            schema: {
              type: 'object',
              required: ['contact', 'activityTitle'],
              properties: {
                "warning": {
                  "type": "string",
                  "description": "No contact found. Enter a name to have a placeholder contact made for you.",
                },
                "contact": {
                  "title": "Contact",
                  "type": "string",
                  "oneOf": [
                    {
                      "const": "xxx",
                      "title": "John Doe",
                      "description": "Candidate - 347",
                    },
                    {
                      "const": "newEntity",
                      "title": "Create placeholder contact"
                    }
                  ],
                },
                "contactName": {
                  "type": 'string',
                  "title": "Contact name",
                },
                "contactType": {
                  "title": "Contact type",
                  "type": "string",
                  "oneOf": [
                    {
                      "const": "candidate",
                      "title": "Candidate"
                    },
                    {
                      "const": "contact",
                      "title": "Contact"
                    }
                  ],
                },
                "activityTitle": {
                  "type": "string",
                  "title": "Activity title"
                },
                "note": {
                  "type": "string",
                  "title": "Note"
                },
              },
            },
            uiSchema: {
              warning: {
                "ui:field": "admonition", // or typography to show raw text
                "ui:severity": "warning", // "warning", "info", "error", "success"
              },
              contactName: {
                "ui:placeholder": 'Enter name',
                "ui:widget": "hidden", // remove this line to show contactName input
              },
              contactType: {
                "ui:placeholder": 'Select contact type',
                "ui:widget": "hidden", // remove this line to show contactType input
              },
              note: {
                "ui:placeholder": 'Enter note',
                "ui:widget": "textarea", // show note input as textarea
              },
              submitButtonOptions: {
                submitText: 'Save',
              },
            },
            formData: {
              contact: 'xxx',
              contactName: '',
              contactType: '',
              activityTitle: 'Outbound call to ...',
              note: '',
            },
          },
        }, '*');
        // navigate to call log page
```

```
        document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
          type: 'rc-adapter-navigate-to',
          path: `/log/call/${data.body.call.sessionId}`,
        }, '*');
      }
      if (data.body.triggerType === 'logForm' || data.body.triggerType === 'presenceUpdate') {
        // Save call log to your platform
        console.log(data.body); // data.body.call, data.body.formData
      }
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
          type: 'rc-post-message-response',
          responseId: data.requestId,
          response: { data: 'ok' },
        }, '*');
      return;
    }
    if (data.path === '/callLogger/inputChanged') {
      console.log(data); // get input changed data in here: data.body.formData
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
      // you can update call log page data here to make the form dynamic
      return;
    }
  }
});
```



3.6.2 Custom page

2.0.0

RingCentral Embeddable is a powerful tool that allows you to customize the user experience for your users. You can create customized pages or tabs to display your own content in the widget.

Register a page

Register a customized page:

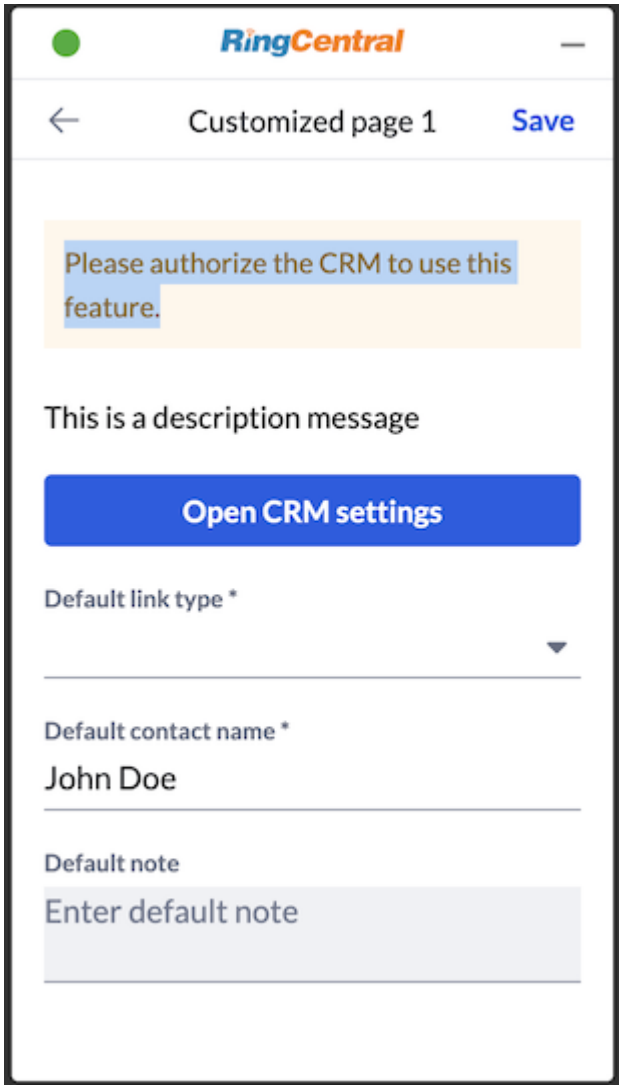
```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-customized-page',
  page: {
    id: 'page1', // page id, required
    title: 'Customized page 1',
    type: 'page',
    // schema and uiSchema are used to customize page, api is the same as [react-jsonschema-form](https://rjsf-team.github.io/react-jsonschema-form)
    schema: {
      type: 'object',
      required: ['contactType', 'defaultContactName'],
      properties: {
        "warning": {
          "type": "string",
          "description": "Please authorize the CRM to use this feature."
        },
        "someMessage": {
          "type": "string",
          "description": "This is a description message"
        },
        "someLink": {
          "type": "string",
          "description": "This is a link message"
        },
        "openSettingsButton": {
          "type": "string",
          "title": "Open CRM settings",
        },
        "contactType": {
          "type": "string",
          "title": "Default link type",
          "oneOf": [
            {
              "const": "candidate",
              "title": "Candidate"
            },
            {
              "const": "contact",
              "title": "Contact"
            }
          ],
        },
        "defaultContactName": {
          "type": "string",
          "title": "Default contact name",
        },
        "defaultNote": {
          "type": "string",
          "title": "Default note",
        },
      },
    },
    uiSchema: {
      submitButtonOptions: { // optional if you don't want to show submit button
        submitText: 'Save',
      },
      warning: {
        "ui:field": "admonition",
        "ui:severity": "warning",  // "warning", "info", "error", "success"
      },
      someMessage: {
        "ui:field": "typography",
        "ui:variant": "body1", // "caption1", "caption2", "body1", "body2", "subheading2", "subheading1", "title2", "title1"
        // "ui:bulletedList": true, // show text as list item // supported from v2.0.1
      },
      someLink: {
        "ui:field": "link", // supported from v2.0.1
        "ui:variant": "body1",
        "ui:color": "avatar.brass",
        "ui:underline": false,
        "ui:href": "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/",
      },
      openSettingsButton: {
        "ui:field": "button",
        "ui:variant": "contained", // "text", "outlined", "contained", "plain"
        "ui:fullWidth": true
```

```
      },
      defaultContactName: {
        "ui:placeholder": 'Enter default contact name',
      },
      defaultNote: {
        "ui:placeholder": 'Enter default note',
        "ui:widget": "textarea", // show note input as textarea
      },
    },
    formData: {
      contactType: 'candidate',
      defaultContactName: 'John Doe',
      defaultNote: '',
    },
  },
}, '*');
```

To update the page, you can re-register the page with new data and same page id.

Navigate to the page:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-navigate-to',
  path: '/customized/page1', // page id
}, '*');
```



Handle button clicked and input changed event

Pass `buttonEventPath` and `customizedPageInputChangedEventPath` when you register service:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    customizedPageInputChangedEventPath: '/customizedPage/inputChanged',
    buttonEventPath: '/button-click',
  }
}, '*');
```

Add event listener to get button clicked and input changed event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/customizedPage/inputChanged') {
      console.log(data); // get input changed data in here: data.body.input
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
      // you can re-register page data here to update the page
      return;
    }
    if (data.path === '/button-click') {
      if (data.body.button.id === 'page1') {
        // on submit button click
        // button id is the page id
        console.log('Save button clicked');
        // ...
      }
      if (data.body.button.id === 'openSettingsButton') {
        // click on the button registered in schema, button id is the button key
        console.log('Open settings button clicked');
        // ...
      }
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

When the user clicks the button, you will receive a message with the path `/button-click`. When the user changes the input, you will receive a message with the path `/customizedPage/inputChanged`.

3.6.3 Custom third-party settings and preferences

This feature requires you to register your app as a service first.

For some features that support user to customize, widget supports to add settings into widget's setting page.

Register settings

First, register service with `settings` and `settingsPath`:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    settingsPath: '/settings',
    settings: [
      {
        "id": "openContactPageAtCall",
        "type": "boolean",
        "name": "Open Contact Page at Call",
        "value": true
      },
    ],
  }
}, '*');
```

REGISTER BUTTON, SECTION AND GROUP

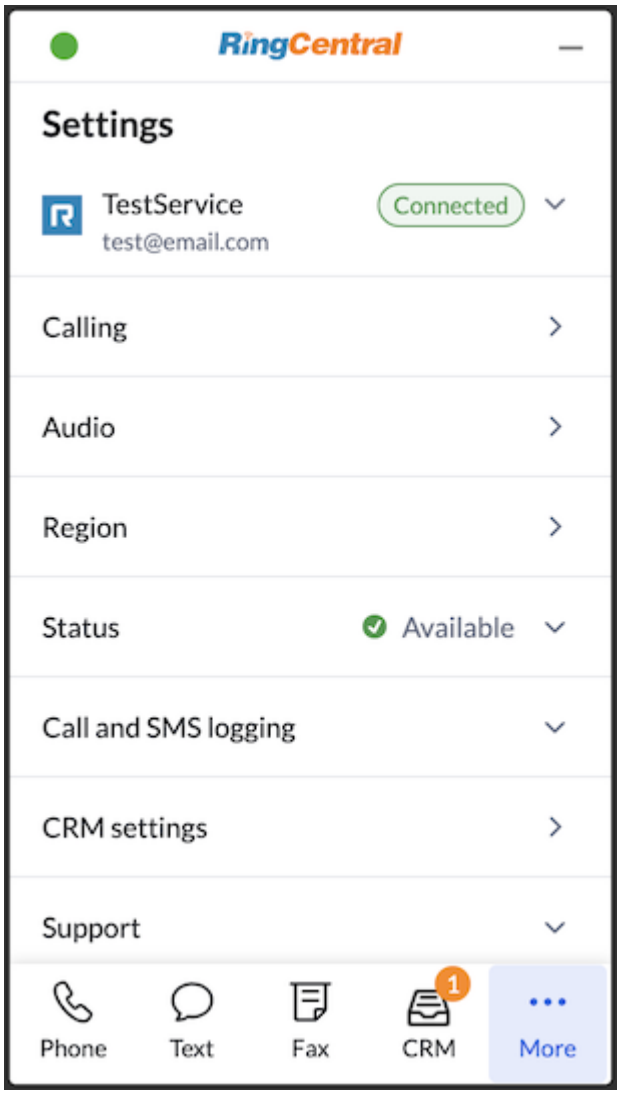
2.0.0

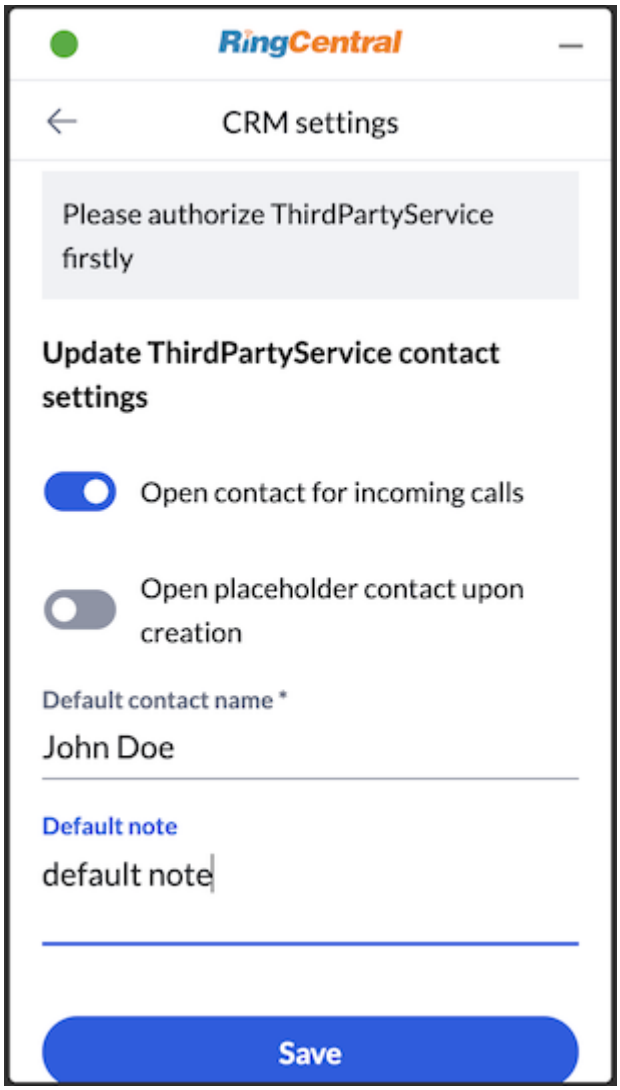
```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    settingsPath: '/settings',
    settings: [
      {
        "id": 'openLoggingPageAfterCall',
        "type": 'boolean',
        "name": 'Open call logging page after call',
        "value": true,
        "groupId": 'logging', // optional, group settings into call and sms logging settings
        // "readOnly": true, // supported from v2.1.0
        // "readOnlyReason": "This setting is managed by admin", // supported from v2.1.0
      },
      {
        "id": "goToAppSettings",
        "type": "button",
        "name": "Go to App settings",
        "buttonLabel": "Open",
      },
      {
        "id": "crmSetting",
        "type": "section",
        "name": "CRM settings",
        "items": [
          {
            "id": "info",
            "name": "info",
            "type": "admonition",
            "severity": "info",
            "value": "Please authorize ThirdPartyService firstly",
          },
          {
            "id": "introduction",
            "name": "Introduction",
            "type": "typography",
            "variant": "body2", // optional, default is body1
            "value": "Update ThirdPartyService contact settings",
          },
          {
            "id": 'openContactPageAtCall',
            "type": 'boolean',
            "name": 'Open contact for incoming calls',
            "value": true,
          },
          {
            "id": "defaultRecordType",
            "type": "option",
            "name": "Default record type",
            "options": [{
```

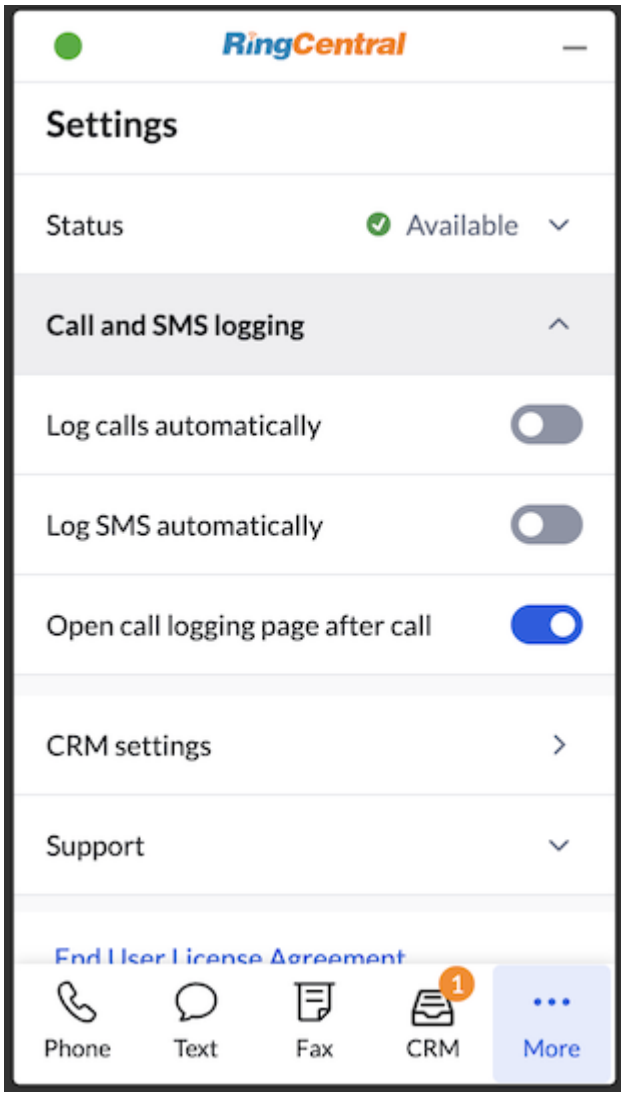
```
              "id": "Lead",
              "name": "Lead"
            }, {
              "id": "Contact",
              "name": "Contact"
            }],
            "value": "",
            "required": true,
            "placeholder": "Select default record type"
          },
          {
            "id": "defaultContactName",
            "type": "string",
            "name": "Default contact name",
            "value": "John Doe",
            "required": true,
            "placeholder": "Input default contact name"
          },
          {
            "id": "defaultNote",
            "type": "text",
            "name": "Default note",
            "value": "",
            "placeholder": "Input default note"
          },
        ]
      },
      {
        "id": "support",
        "type": "group",
        "name": "Support",
        "items": [{
          "id": "document",
          "type": "externalLink",
          "name": "Document",
          "uri": "https://www.google.com",
        }, {
          "id": "feedback",
          "type": "button",
          "name": "Feedback",
          "buttonLabel": "Open",
          "buttonType": "link",
        }, {
          "id": "devSupport",
          "type": "button",
          "name": "Developer support",
          "buttonLabel": "Open",
        }]
      },
    ],
    buttonEventPath: '/button-click', // required if you have button type in settings
  }
}, '*');
```

> **In settings root items, it only supports `boolean`, `button`, `section` and `group` type. In section's items, it supports `boolean`, `string`, `option`, `text`, `typography` and `admonition` type.**

After registering, you can get your setting in settings page:







Add a message event to listen settings updated event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/settings') {
      // add your codes here to save settings into your service
      console.log(data);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
    if (data.path === '/button-click') {
      // add your codes here to handle button click event
      console.log(data);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

LISTEN FOR SETTING BUTTON CLICK

2.0.0

```
window.addEventListener('message', function (e) {
  var data = e.data;
```

```
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/button-click') {
      // add your codes here to handle button click event
      console.log(data);
      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

Set settings item order

2.0.0

You can set settings item order by adding `order` field in settings item:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    settingsPath: '/settings',
    settings: [
      {
        "id": "settingItem1",
        "type": "button",
        "name": "Setting Item 1",
        "buttonLabel": "Open",
        "order": 250 // the smaller the number, the higher the priority.
        // Calling setting order value: 100,
        // Audio setting order value: 200,
        // Region setting order value: 300,
        // Status setting order value: 400,
        // Call and SMS logging setting order value: 500,
      },
    ],
    buttonEventPath: '/button-click', // required if you have button type in settings
    // ...
  }
}, '*');
```

Update settings

2.0.0

You can update settings by sending `rc-adapter-update-third-party-settings` message:

```
  document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
    type: 'rc-adapter-update-third-party-settings',
    settings: [
      {
        "id": "openContactPageAtCall",
        "type": "boolean",
        "name": "Open Contact Page at Call",
        "value": true,
      },
      {
        "id": "openCRMPage",
        "type": "button",
        "name": "Go to app settings",
        "buttonLabel": "Open",
        "order": 10000,
      },
    ],
  }, '*');
```

3.6.4 Custom tab

2.0.0

RingCentral Embeddable is a powerful tool that allows you to customize the user experience for your users. You can create customized pages or tabs to display your own content in the widget.

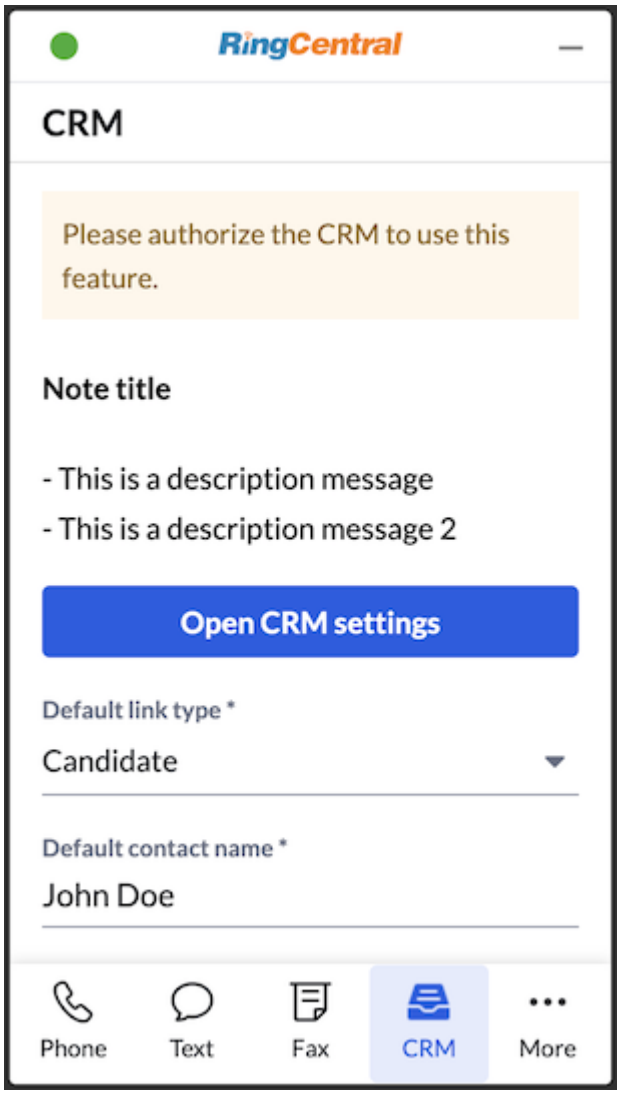
Register a tab

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-customized-page',
  page: {
    id: 'tabID', // tab id, required
    title: 'CRM',
    type: 'tab', // tab type
    iconUri: 'https://xxx/icon.png', // icon for tab, 24x24, recommended color: #16181D
    activeIconUri: 'https://xxx/icon-active.png', // icon for tab in active status, 24x24, recommended color: ##2559E4
    darkIconUri: 'https://xxx/icon-dark.png', // Supported from v2.2.1, icon for tab in dark mode, 24x24, recommended color: #ffffff
    hidden: false, // optional, default false, whether to hide the tab icon from navigation bar
    unreadCount: 0, // optional, unread count, 0-99
    priority: 31, // tab priority, 0-100, 0 is the highest priority, Phone tab: 10, Text: 20, Fax: 30, Glip: 40, Contacts: 50, Video: 60, Settings: 70
    // schema and uiSchema are used to customize page, api is the same as [react-jsonschema-form](https://rjsf-team.github.io/react-jsonschema-form)
    schema: {
      type: 'object',
      required: ['contactType', 'defaultContactName'],
      properties: {
        "warning": {
          "type": "string",
          "description": "Please authorize the CRM to use this feature."
        },
        "someMessage": {
          "type": "string",
          "description": "This is a description message"
        },
        "openSettingsButton": {
          "type": "string",
          "title": "Open CRM settings",
        },
        "contactType": {
          "type": "string",
          "title": "Default link type",
          "oneOf": [
            {
              "const": "candidate",
              "title": "Candidate"
            },
            {
              "const": "contact",
              "title": "Contact"
            }
          ],
        },
        "defaultContactName": {
          "type": "string",
          "title": "Default contact name",
        },
        "defaultNote": {
          "type": "string",
          "title": "Default note",
        },
      },
    },
    uiSchema: {
      submitButtonOptions: { // optional if you don't want to show submit button
        submitText: 'Save',
      },
      warning: {
        "ui:field": "admonition",
        "ui:severity": "warning",  // "warning", "info", "error", "success"
      },
      someMessage: {
        "ui:field": "typography",
        "ui:variant": "body1", // "caption1", "caption2", "body1", "body2", "subheading2", "subheading1", "title2", "title1"
      },
      openSettingsButton: {
        "ui:field": "button",
        "ui:variant": "contained", // "text", "outlined", "contained", "plain"
        "ui:fullWidth": true
      },
      defaultContactName: {
        "ui:placeholder": 'Enter default contact name',
      },
      defaultNote: {
        "ui:placeholder": 'Enter default note',
        "ui:widget": "textarea", // show note input as textarea
      },
```

```
    },
    formData: {
      contactType: 'candidate',
      defaultContactName: 'John Doe',
      defaultNote: '',
    },
  },
}, '*');
```

Navigate to the tab:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-navigate-to',
  path: '/customizedTabs/tabID', // page id
}, '*');
```



Handle button clicked and input changed event

Pass `buttonEventPath` and `customizedPageInputChangedEventPath` when you register service:

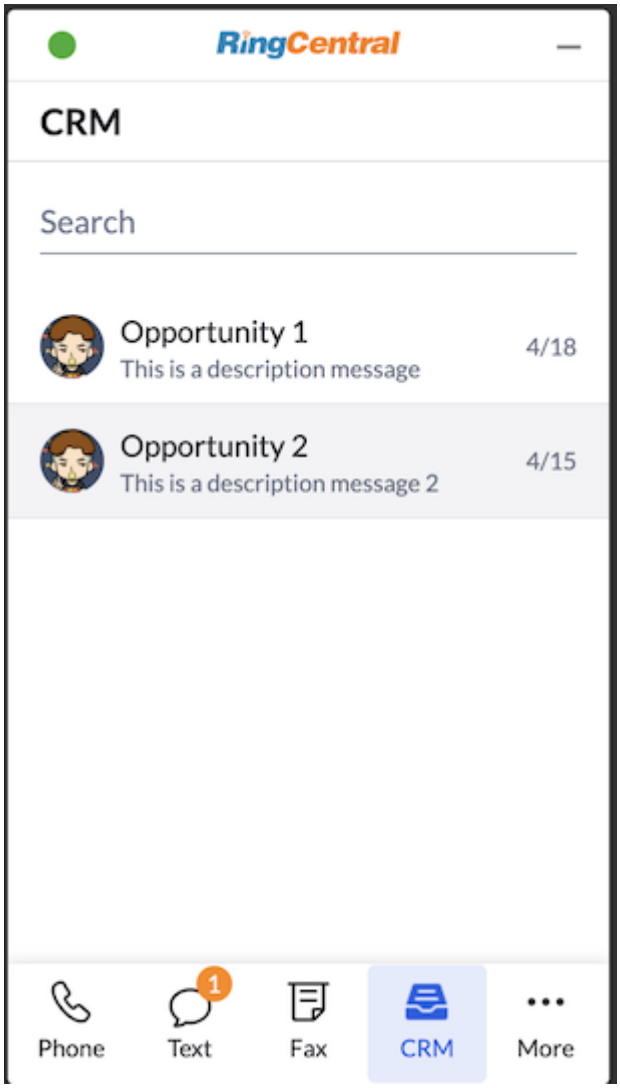
```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    customizedPageInputChangedEventPath: '/customizedPage/inputChanged',
    buttonEventPath: '/button-click',
  }
}, '*');
```

Add event listener to get button clicked and input changed event:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/customizedPage/inputChanged') {
      console.log(data); // get input changed data in here: data.body.input
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
      // you can re-register page data here to update the page
      return;
    }
    if (data.path === '/button-click') {
      if (data.body.button.id === 'page1') {
        // on submit button click
        // button id is the page id
        console.log('Save button clicked');
        // ...
      }
      if (data.body.button.id === 'openSettingsButton') {
        // click on the button registered in schema, button id is the button key
        console.log('Open settings button clicked');
        // ...
      }
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
        response: { data: 'ok' },
      }, '*');
    }
  }
});
```

When the user clicks the button, you will receive a message with the path `/button-click`. When the user changes the input, you will receive a message with the path `/customizedPage/inputChanged`.

Show list in customized tab



You can show a list in the customized page:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-customized-page',
  page: {
    id: 'tabID', // tab id, required
    title: 'CRM',
    type: 'tab', // tab type
    iconUri: 'https://xxx/icon.png', // icon for tab, 24x24
    activeIconUri: 'https://xxx/icon-active.png', // icon for tab in active status, 24x24
    priority: 31,
    // schema and uiSchema are used to customize page, api is the same as [react-jsonschema-form](https://rjsf-team.github.io/react-jsonschema-form)
    schema: {
      type: 'object',
      required: [],
      properties: {
        "search": {
          "type": "string",
        },
        "opportunity": {
          "type": "string",
          "oneOf": [{
            "const": "opportunity1",
            "title": "Opportunity 1",
            "description": "This is a description message",
            "meta": "4/18",
            "icon": "https://xxx/icon1.png"
          }, {
            "const": "opportunity2",
            "title": "Opportunity 2",
            "description": "This is a description message 2",
            "meta": "4/15",
            "icon": "https://xxx/icon1.png"
          }]
```

```
        },
        "section": {
          "type": "string",
          "oneOf": [{
            "const": "advanced1",
            "title": "Advanced settings 1",
          }, {
            "const": "advanced2",
            "title": "Advanced settings 2",
          }]
        },
      },
    },
    uiSchema: {
      search: {
        "ui:placeholder": 'Search',
        "ui:label": false,
      },
      opportunity: {
        "ui:field": "list",
        "ui:showIconAsAvatar": true, // optional, default true. show icon as avatar (round) in list
      },
      section: {
        "ui:field": "list",
        "ui:navigation": true, // optional, default false. show list as navigation items, supported from v2.1.0. it can be used to navigate to another page
      },
    },
    formData: {
      search: '',
      opportunity: '',
    },
  },
}, '*');
```

When user clicks on the list item, you will receive a message with the path `/customizedPage/inputChanged`.

3.7 Recipes

3.7.1 Enable analytics

Developers can implement their own custom event tracking with internal or third-party analytics systems using Embeddable's API. This feature is disabled by default. To enable analytics tracking, enable the `enableAnalytics` configuration parameter.

Listen for the track event

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-analytics-track':
        // get analytics data
        console.log('rc-analytics-track:', data.event, data.properties);
        break;
      default:
        break;
    }
  }
});
```

3.7.2 Working with RingCentral's click-to-dial library

This is document that show how to implement `Click To Dial` feature with RingCentral C2D library. RingCentral C2D is a library that help developers to implement `Click To Dial` and `Click To SMS` feature, it will scan phone numbers in web page. When users hover on phone number, it will show C2D widget for `Click to Call`.

To implement with RingCentral Embeddable:

```
<script src="https://unpkg.com/ringcentral-c2d@1.0.0/build/index.js"></script>
<script>
// Inject Embeddable
(function() {
  var rcs = document.createElement("script");
  rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js";
  var rcs0 = document.getElementsByTagName("script")[0];
  rcs0.parentNode.insertBefore(rcs, rcs0);
})();
// Interact with RingCentral C2D
var clickToDial = new RingCentralC2D();
clickToDial.on(RingCentralC2D.events.call, (phoneNumber) => {
  RCAdapter.clickToCall(phoneNumber, true);
});
clickToDial.on(RingCentralC2D.events.text, (phoneNumber) => {
  RCAdapter.clickToSMS(phoneNumber);
});
</script>
```

3.7.3 Call pop

This page describes how to implement the call pop feature based on the Embeddable events.

Listen for the active call event

```
window.addEventListener('message', (e) => {
  const data = e.data;
  if (data) {
    switch (data.type) {
      case 'rc-active-call-notify':
        // only pop call for incoming ringing call
        if (data.call.direction === 'Inbound' && data.call.telephonyStatus === 'Ringing') {
          // here we popup a Google form pre-fill uri:
          const formUri = `https://docs.google.com/forms/d/e/xxxxxxxxx/viewform?usp=pp_url&entry.985526131=${data.call.direction}&entry.1491856435=$
{data.call.from.phoneNumber}&entry.875629840=${encodeURIComponent(data.call.fromName)}&entry.1789287962=${data.call.to.phoneNumber}&entry.1281736933=$
{encodeURIComponent(data.call.toName)}`;
          window.open(formUri, 'Call form', 'width=600,height=600');
        }
        break;
      default:
        break;
    }
  }
});
```

Here we listen to active call event. When there is an incoming call, it will popup a Google Forms pre-fill uri. Get the online demo here.

3.7.4 Add feedback prompt in Settings tab

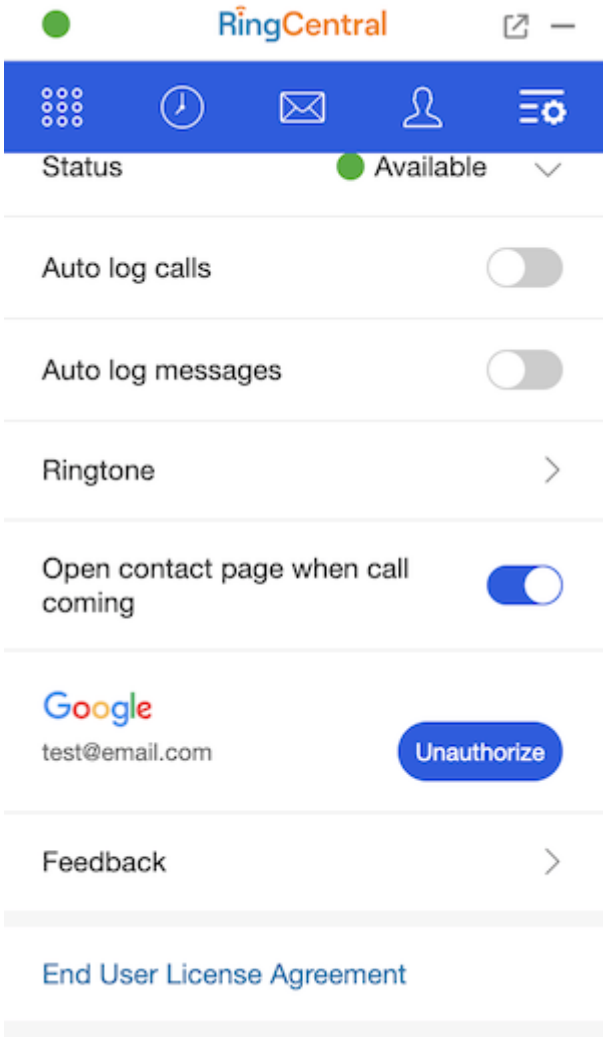
> **This feature requires you to register your app as a service first.**

For developer who want to add feedback feature, the app provides a API to show a feed link in settings page:

First, register service with `feedbackPath`:

```
document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
  type: 'rc-adapter-register-third-party-service',
  service: {
    name: 'TestService',
    feedbackPath: '/feedback',
  }
}, '*');
```

After registering, you can get feedback link in settings page:



Add a message event to listen feedback link click event and handle that:

```
window.addEventListener('message', function (e) {
  var data = e.data;
  if (data && data.type === 'rc-post-message-request') {
    if (data.path === '/feedback') {

      // response to widget
      document.querySelector("#rc-widget-adapter-frame").contentWindow.postMessage({
        type: 'rc-post-message-response',
        responseId: data.requestId,
```

```
        response: { data: 'ok' },
      }, '*');
      // add your codes here to show your feedback form
      console.log(data);
    }
  }
});
```

Add feedback button at header

You can also add a feedback button at header, this way doesn't require to register service:

1.10.0

```
RCAdapter.showFeedback({
  onFeedback: function () {
    // add your codes here to show your feedback form
  },
});
```

3.8 Migration

3.8.1 Migrating from Github Page latest URI

In previously, we deployed the latest build at Github Page: `https://ringcentral.github.io/ringcentral-embeddable/` . And now the latest build is deployed at `https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/` to have more stable network access.

To migrate to the new latest URI, you can just replace the old URI with the new one.

Javascript **iframe**

Update adapter js src:

```
<script>
  (function() {
    var rcs = document.createElement("script");
    rcs.src = "https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/adapter.js";
    var rcs0 = document.getElementsByTagName("script")[0];
    rcs0.parentNode.insertBefore(rcs, rcs0);
  })();
</script>
```

Update iframe src:

```
<iframe width="300" height="500" allow="microphone"
  src="https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/app.html">
</iframe>
```

Then **add** new redirect URI in your RingCentral app settings to

```
https://apps.ringcentral.com/integration/ringcentral-embeddable/latest/redirect.html
```

After migrating, user will need to **re-authorize** RingCentral to your app to use the widget as domain changed.

4. Support and troubleshooting

4.1 Limited support on non-supported browsers

RingCentral Embeddable is full tested on the latest versions of the following browsers:

- Google Chrome
- Microsoft Edge (a Chromium-based browser)
- Mozilla Firefox

In all other browsers, support may be limited, especially browsers that do not support WebRTC. For those browsers you may see a warning indicating that the web phone is "unavailable." For these browsers, you can modify your Calling settings and set your calling mode to either:

- Call with RingCentral Desktop app
- RingOut

4.2 Granting access to speakers and microphones

RingCentral Embeddable requests the `userMedia` permission from your browser in order to access a computer's microphone and speaker, necessary for making phone calls. There are three circumstances that will cause this request to fail:

- There is no microphone and speaker devices in your computer
- The end user does not click "Allow" when the request is made
- The widget is not served via HTTPS

In Chrome or Firefox, browsers will block the `userMedia` request for a non-HTTPS website. One can forcibly circumvent this limitation in one of the following ways (this is not recommended for production use):

- Chrome users: goto `chrome://flags/#unsafely-treat-insecure-origin-as-secure` page, enable `Insecure origins treated as secure`
- Firefox users: goto `about:config` page, enable `media.getusermedia.insecure`

4.3 Enabling active call control features

Active Call Control feature uses new `CallControl` RESTful API to control RingCentral Call. With this API, users can control their calls on other devices in this widget.

Before we start to use Active Call Control feature, need to add `CallControl` permission to your app in RingCentral Developer website. After permissions added, you can get the feature after re-login to the widget.

Please submit a help ticket to add `CallControl` permission if you get any problem.

4.4 Enabling the conference calling feature

To enable Conference Call (3-way-calling) feature, please add `CallControl` permission to your app via the RingCentral Developer console. If your app has already been graduated and you need help, please submit a help ticket and we can assist. After the permission is added you can obtain the feature after you re-login to the widget.