RingCentral Web Phone 2

This is a complete rewrite of the RingCentral Web Phone SDK.

It is NOT yet producion ready. It is still in development.

Demo

- Online Demo
- Source Code

Pre-requisites

This SDK assumes that you have basic knowledge of RingCentral Platform. You have created a RingCentral app and you know how to invoke RingCentral APIs. If you don't know how to do that, please read the following document first: https://developers.ringcentral.com/guide/voice/call-log/quick-start. The document is about how to create a RingCentral app and how to use the RingCentral API to access call log data. It is a good starting point for you to understand the RingCentral API. This SDK doesn't use/require call log API, the document is just for you to get familiar with RingCentral API.

This SDK assumes that you know how to invoke Device SIP Registration to get a `sipInfo` object.

With `@ringcentral/sdk`, it is done like this:

```
import { SDK } from '@ringcentral/sdk';

const rc = new SDK({
  server: process.env.RINGCENTRAL_SERVER_URL,
  clientId: process.env.RINGCENTRAL_CLIENT_ID,
  clientSecret: process.env.RINGCENTRAL_CLIENT_SECRET,
});

const main = async () => {
  await rc.login({
    jwt: process.env.RINGCENTRAL_JWT_TOKEN,
  });
  const r = await rc.platform().post('/restapi/v1.0/client-info/sip-pro
    sipInfo: [{ transport: 'WSS' }],
  });
  const jsonData = await r.json();
  const sipInfo = jsonData.sipInfo[0];
  console.log(sipInfo); // this is what we need

  const deviceId = jsonData.device.id; // Web Phone SDK doesn't need `d
  await rc.logout(); // Web Phone SDK doesn't need a long-living Restfu
};
main();
```

With `@rc-ex/core`, it is done like this:

```
import RingCentral from '@rc-ex/core';

const rc = new RingCentral({
  server: process.env.RINGCENTRAL_SERVER_URL,
  clientId: process.env.RINGCENTRAL_CLIENT_ID,
  clientSecret: process.env.RINGCENTRAL_CLIENT_SECRET,
});

const main = async () => {
  await rc.authorize({
    jwt: process.env.RINGCENTRAL_JWT_TOKEN!,
  });
  const r = await rc
    .restapi()
    .clientInfo()
    .sipProvision()
    .post({
      sipInfo: [{ transport: 'WSS' }],
    });
  const sipInfo = r.sipInfo![0];
  console.log(sipInfo); // this is what we need
```

```
  const deviceId = r.device!.id; // Web Phone SDK doesn't need `deviceI
  await rc.revoke(); // Web Phone SDK doesn't need a long-living Restfu
};
main();
```

Please note that, you may save and re-use `sipInfo` for a long time. You don't need to invoke `Device SIP Registration` every time you start the web phone.

In the sample code above, I also showed you how to get the `deviceId`. Web Phone SDK doesn't need `deviceId`, it is just for your information. Just in case you may need it for RingCentral Call Control API.

Installation

```
yarn add ringcentral-web-phone@2.0.0-alpha.1
```

At the time I am writing this document, the latest version is `2.0.0-alpha.1`. Please replace it with the latest version. Find the latest version here https://www.npmjs.com/package/ringcentral-web-phone

Initialization

```
import WebPhone from 'ringcentral-web-phone';

const webPhone = new WebPhone({ sipInfo });
await webPhone.register();
```

What is `sipInfo`? Please read Pre-requisites section.

Optionally, you can specify `instanceId`: `new WebPhone({ sipInfo, instanceId })`. `instanceId` is the unique ID of your web phone device.

If you want like to run multiple web phone devices in multiple tabs, you need to generate a unique `instanceId` for each device. It MUST be persistent across power cycles of the device. It MUST NOT change as the device moves from one network to another. Ref: https://datatracker.ietf.org/doc/html/rfc5626#section-4.1

If you start two web phone instances with the same `instanceId`, only the second instance will work. SIP server will not route calls to the first instance. (The first instance will still be able to make outbound calls, but it will not receive inbound calls.)

If you don't specify `instanceId`, the SDK by default will use `sipInfo.authorizationId` as `instanceId`. Which means, if you don't specify `instanceId`, you should only run one web phone instance in one tab.

If you start two web phone instances with different `instanceId`, both instances will work. SIP server will send messages to both instances.

Debug Mode

```
await webPhone.enableDebugMode();
```

In debug mode, the SDK will print all SIP messages to the console. It is useful for debugging.

Make an outbound call

```
const callSession = await webPhone.call(callee, callerId);
```

`callee` is the phone number you want to call. Format is like `16506668888`. `callerId` is the phone number you want to display on the callee's phone. Format is like `16506668888`.

To get all the `callerId` that you can use, you can call the following API: https://developers.ringcentral.com/api-reference/Phone-Numbers/listExtensionPhoneNumbers. Don't forget to filter the phone numbers that have `"features": [..., "CallerId", ...]`.

Get inbound call sessions

To get inbound call sessions, you can listen to the `inboundCall` event:

```
webPhone.on('inboundCall', (inbundCallSession: InboundCallSession) => {
  // do something with the inbound call session
});
```

Actions to take on inbound call session

Answer the call

```
await inbundCallSession.answer();
```

Decline the call

```
await inbundCallSession.decline();
```

Please note that, decline the inbound call will not terminate the call session for the caller immediately. The caller will hear the ringback tone for a while until he/she hears "I am sorry, no one is available to take your call. Thank you for calling. Goodbye." And the call will not reach your voicemail.

Send the call to voicemail

```
await inbundCallSession.toVoicemail();
```

Forward the call

```
await inbundCallSession.forward(targetNumber);
```

Reply the call

Optionally, you can tell the server that the user has started replying the call. The server will give the user more time to edit the reply message before ending the call or redirecting the call to voicemail.

```
await inbundCallSession.startReply();
```

Reply the call with text:

```
const response = await inbundCallSession.reply(text);
```

After this method call, the call session will be ended for the callee. But the call session will not end yet for the caller. And the caller will receive the replied `text` via text-to-speech. The caller will then have several options:

- press 1 to repeat the message
- press 2 to leave a voicemail

- press 3 to reply with "yes"
- press 4 to reply with "no"
- press 5 to reply with "urgent, please call immediately"
  - the caller will be prompted to specify a callback number
- press 6 to to disconnect

`if (response.body.Sts === '0')`, it means that the caller replied to your message(he/she pressed 3, 4, 5). Then you need to check `response.body.Resp`:

- if it's `'1'`, it means that the caller replied with "yes" (he/she pressed 3)
- if it's `'2'`, it means that the caller replied with "no" (he/she pressed 4)
- if it's `'3'`, it means that the caller replied with "urgent, please call [number] immediately". (he/she pressed 5)
  - in this case, there is also an urgent number provided by the caller which can be accessed by `response.body.ExtNfo`.

Below is some code snippet for your reference:

```
const response = await session.reply('I am busy now, can I call you bacl
if (response.body.Sts === '0') {
  const message = `${response.body.Phn} ${response.body.Nm}`;
  let description = '';
  switch (response.body.Resp) {
    case '1':
      description = 'Yes';
      break;
    case '2':
      description = 'No';
      break;
    case '3':
      description = `Urgent, please call ${response.body.ExtNfo} immedi
      break;
    default:
      break;
  }
  global.notifier.info({
    message, // who replied
    description, // what replied
    duration: 0,
  });
}
```

Actions to take on answered call sessions

This part applies to both inbound and outbound call sessions. Once the call is answered, you can do the following actions:

Transfer the call

"Cold" transfer

It is also called blind transfer. Transfer the call to another number directly, without any introduction or context to the person to whom the call will be transferred (the transferee).

```
await callSession.transfer(targetNumber);
```

"Warm" transfer

The original caller is placed on hold while the person handling the call (the transferor) speaks with the person to whom the call will be transferred (the transferee). The transferor introduces the caller, provides context, and confirms that the transferee is ready to take the call before connecting the two.

```
const { complete, cancel } = await session.warmTransfer(transferToNumbe
```

After this method call, the current call session will be put on hold. A new call session will be created to the `transferToNumber`. Then the transferor will have a chance to talk to the transferee. After that, depending on the transferor's decision, the app can call `complete()` to complete the transfer, or call `cancel()` to cancel the transfer.

Hang up the call

```
await callSession.hangup();
```

Start/Stop call recording

```
await callSession.startRecording();
await callSession.stopRecording();
```

Flip the call

```
const result = await callSession.flip(targetNumber);
```

Most popular use case of call flip is for you to switch the current call to your other devices. Let's say you are talking to someone on your desktop, and you want to switch to your mobile phone. You can use call flip to achieve this: `await callSession.flip(mobilePhoneNumber)`.

Please note that, after you mobile phone answers the call, you need to **manually** end the call session on your desktop, otherwise you won't be able to talk/listen on your mobile phone.

Please also note that, this SDK allows you to flip the call to any phone number, not just your own phone numbers. But if it is not your number, you probably should transfer the call instead of flipping the call.

A sample result of `flip` is like this:

```
{
  "code": 0,
  "description": "Succeeded",
  "number": "+16506668888",
  "target": "16506668888"
}
```

I don't think you need to do anything based on the result. It is just for your information.

Park the call

```
const result = await callSession.park();
```

After this method call, the call session will be ended for you. And the remote peer will be put on hold and parked on an extension. You will be able to retrieve the parked call by dailing `*[parked-extension]`. Sample result:

```
{
  "code": 0,
  "description": "Succeeded",
  "park extension": "813"
}
```

Take the sample result above as an example, you can retrieve the parked call by dailing `*813`.

Hold/Unhold the call

```
await callSession.hold();
await callSession.unhold();
```

If you put the call on hold, the remote peer will hear hold music. Neither you nor the remote peer can hear each other. If you unhold the call, you and the remote peer can hear each other again.

Mute/Unmute the call

```
await callSession.mute();
await callSession.unmute();
```

If you mute the call, the remote peer can't hear you. If you unmute the call, the remote peer can hear you again.

Send DTMF

```
await callSession.sendDTMF(dtmf);
```

`dtmf` is a string, like `*123#`. Valid characters are `0123456789*#ABCD`. `ABCD` are less commonly used but are part of the DTMF standard. They were originally intended for special signaling in military and network control systems.

Events

You may subscribe to events, examples:

```
webPhone.on('message', (inboundMessage: InboundMessage) => {
  // do something with the inbound SIP message
});
```

```
callSession.on('disposed', () => {
  // do something when the call session is disposed
```

```
});
```

WebPhone Events

- message
  - new inbound SIP message, payload type: InboundMessage
- inboundCall
  - new inbound call session, payload type: InboundCallSession
- outboundCall
  - new outbound call session, payload type: OutboundCallSession

CallSession Events

- ringing
  - no payload
- answered
  - no payload
- disposed
  - no payload

Conference

Conference is out of the scope of this SDK. Because conferences are mainly done with Restful API. With above being said, I will provide some code snippets for your reference.

Create a conference

To create a conference: https://developers.ringcentral.com/api-reference/Call-Control/createConferenceCallSession If you are using SDK `@rc-ex/core`, you can do it like this:

```
const r = await rc.restapi().account().telephony().conference().post();
```

In the response of the above API call, you will get a `r.session!.voiceCallToken!`. As the host, you will need to dial in:

```
const confSession = await webPhone.call(r.session!.voiceCallToken!);
```

Invite a number to the conference

Make a call to the number you want to invite to the conference:

```
const callSession = await this.webPhone.call(targetNumber);
```

Whenever the call is answered, you can bring in the call to the conference:

```
callSession.once('answered', async () => {
  await rc.restapi().account().telephony().sessions(confSession.session
    sessionId: callSession.sessionId,
    partyId: callSession.partyId,
  });
});
```

Merge an existing ongoing call to the conference

Let's say an existing call session is `callSession`.

```
await rc.restapi().account().telephony().sessions(confSession.sessionId
  sessionId: callSession.sessionId,
  partyId: callSession.partyId,
});
```

You can see that it doesn't matter how the call is created, it could be either an outbound call or an inbound call. You could create it on-the-fly or you can find an existing call session.

A live sample

For a live sample, please refer to https://github.com/tylerlong/rc-web-phone-demo-2

Breaking changes

API changes

2.x version is a complete rewrite of the RingCentral Web Phone SDK. The API is completely different from 1.x version.

Behavior changes

ringing audio

This SDK doesn't play ringing audio when there is incoming call or outgoing call. It's up to the developer/app to play the audio. It's a by design change.

We made this change because we want to give the developer/app more flexibility. And playing ringing audio is not a core feature of the SDK. It's more about how the app interacts with end users.

call forward

SDK 1.x treats forwarding as answering the call and then transfer the call. SDK 2.x treats forwarding as sending a SIP message to the SIP server to forward the call. I would like to say this is more like a bug fix than a behavior change.

Maintainers Notes

Content below is for the maintainers of this project.

References

- ref: https://www.ietf.org/rfc/rfc3261.txt

How to test

rename `.env.sample` to `.env` and fill in the correct values. You will need two RingCentral extensions to test the SDK, one as the caller and the other as the callee. You will need the `sipInfo` json string of the two extensions. Invoke this API to get `sipInfo`.

You will need one more number to test call forwarding/transferring.

To run all tests:

```
yarn test
```

To run a test file:

```
yarn test test/inbound/forward.spec.ts
```

Two kinds of special messages

Before an incoming call is answered, client may send special messages with **XML** body to confirmReceive/toVoicemail/decline/forward/reply the call.

In an ongoing call (either inbound or outbound), client may send special messages with **JSON** body to startCallRecord/stopCallRecord/flip/park the call.

webPhone.sipRegister(0)

Register the SIP client with expires time 0. It means that the SIP client will be unregistered immediately after the registration. After this method call, no incoming call will be received. If you try to make an outbound call, you will get a `SIP/2.0 403 Forbidden` response.

Todo:

- create some slides to talk about the reasoning for getting rid of SIP.js
- How to decouple SIP from WebRTC?
- generate api reference