



Avaya Web Messenger - Development Guide

Release 2.3.2
September 2022

Table of Contents

Introduction	4
Capabilities	5
Getting Started	9
What's included in the bundle?	9
API reference	9
Browser support	10
Installation	11
Step 1: Include the Web Messenger on your web page.....	11
Step 2: Initialize with your new app ID.....	11
Region configuration.....	12
Login with userId and JWT	12
Customization	13
Embedded mode	13
Strings customization.....	13
Date localization.....	14
Display Style	15
Button Size	15
Fixed Intro	15
Colors	16
Business profile.....	16
Background Image	16
Sound notification	17
Browser storage.....	17
Menu items	18
Filtering and transforming messages	18
Whitelisting Domains	21
Content Security Policy	22
Appendix A – Users Authentication	23
Appendix B - Managing Users	27
Appendix C - Web Messenger FAQ.....	29
Appendix D – certificate configuration for file transfer feature.....	31

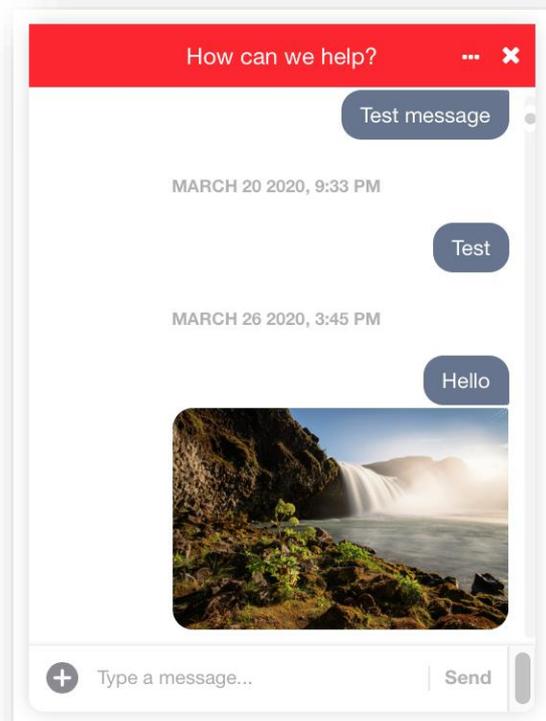


Web Messenger

Current version: v2.3.2

The Avaya Web Messenger is a highly customizable messaging widget that can be added to any web page. It supports the Avaya IX Digital Connection API [capabilities](#).

- Introduction
- Capabilities
- Getting Started
 - SDK bundle
 - API reference
- Browser Support
- Installation
- Region Configuration
- Authenticating users
- Customization
- Filtering and transforming messages
- Whitelisting Domains
- Content Security Policy
- Appendices
 - Appendix A – Users Authentication
 - Appendix B - Managing Users
 - Appendix C - Android FAQ



Introduction

This document is a development guide and contains instructions and information for developers seeking to integrate Web messenger, a highly customizable messaging widget supported by Avaya IX Digital Connection platform.

What is the Avaya IX Digital Connection?

The Avaya IX Digital Connection is a software platform that enables businesses to communicate with their customers across several popular messaging apps.

Developers can use the Avaya IX Digital Connection along with the SDK to add messaging and conversational capabilities to their software. Avaya IX Digital Connection's rich APIs allow for conversation management, rich messaging, user metadata collection, account management and more.

Businesses can also use the Avaya IX Digital Connection to connect to their customers (with agents, bots) over messaging using an Avaya contact center solution.

IMPORTANT NOTE: This Development guide is applicable to Web Messenger release v2.3.2

Who is this for?

- Product teams who want to add web messaging capabilities to their own software.
- Developers who want the richest web messenger available with a powerful, simple, and customizable SDK.
- Bot builders who want to build a web-based bots using a powerful SDK.
- Businesses who want to add chat to their website, and allow the conversation to live beyond the browser and in any messaging channel.
- Customer success teams who want to proactively engage web visitors and build engaging relationships.
- Sales teams who wants to do commerce and upsell from their website.

What you'll need

- Some technical skills and access to your website code.

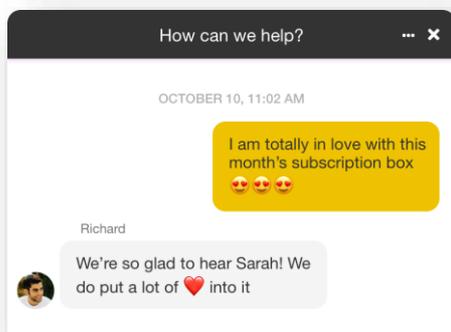
Capabilities

Web Messenger supports a wide variety of capabilities. Below is a detailed view of each capability:

Text and Emoji

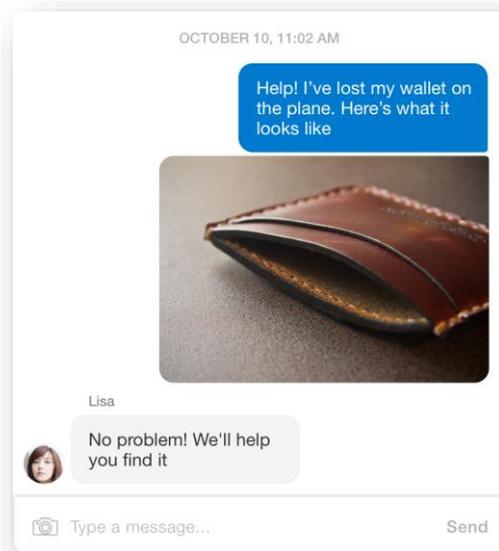
Plain text messages and Unicode Emojis 🌟

Web Messenger displays any Unicode emoji sent in text messages. Mobile users can use the emoji keyboard on their device to send them. Web Messenger does not include a visual emoji selector at the moment.



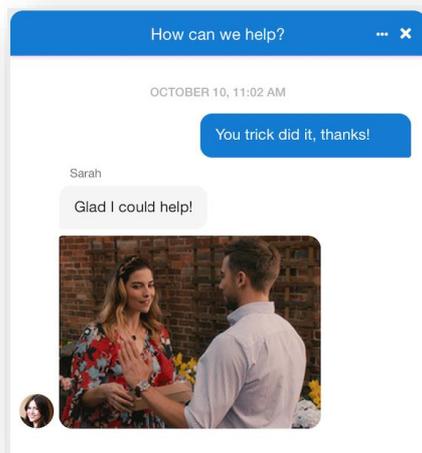
Image

Display static images.



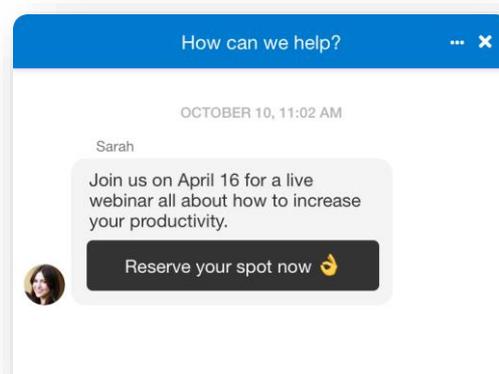
GIF

GIFs sent via both the API and the Web Messenger will be animated. The Web messenger does not include a GIF picker.



Link

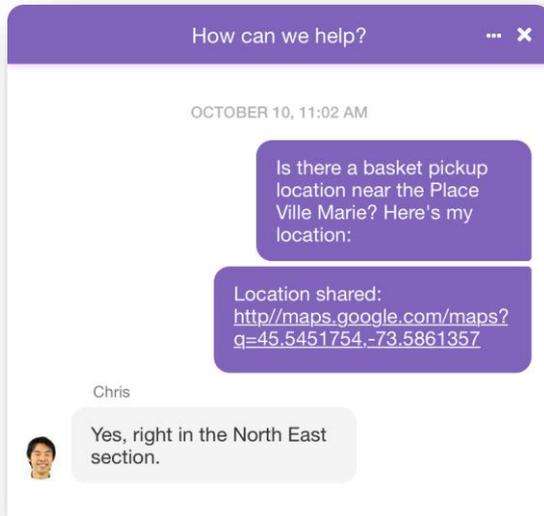
Display web links as buttons, Transform links into clear calls to action.



Location

Send and receive geolocation messages

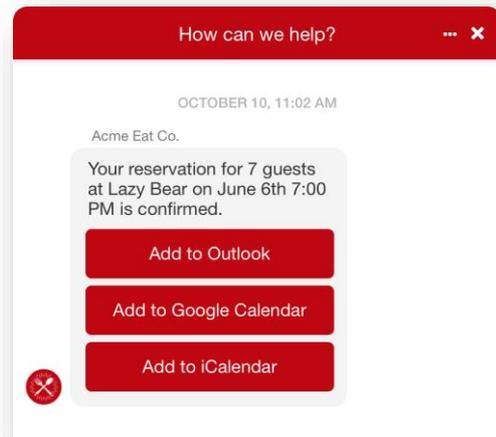
Users will see the message rendered as text with a Google Maps link of the location. Location messages include longitude and latitude coordinates in the API.



Postback

Send buttons to trigger events on your server.

Postback buttons notify the server by webhook when clicked. The server can then act on the click and post messages back to the user in response to the click.



Reply

You can suggest a few answers to reply to a message.

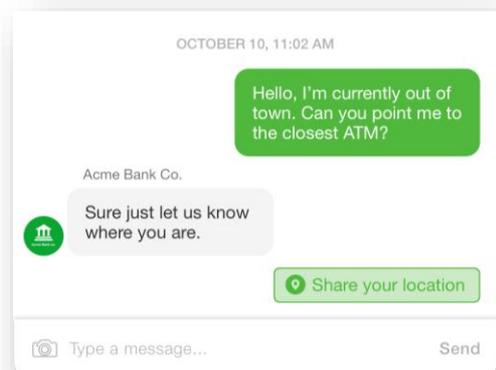
When including replies with a message, Web Messenger will display them at the bottom of the conversation. Users can quickly select one of them to send that reply.



Location Request

Request the current location of the user

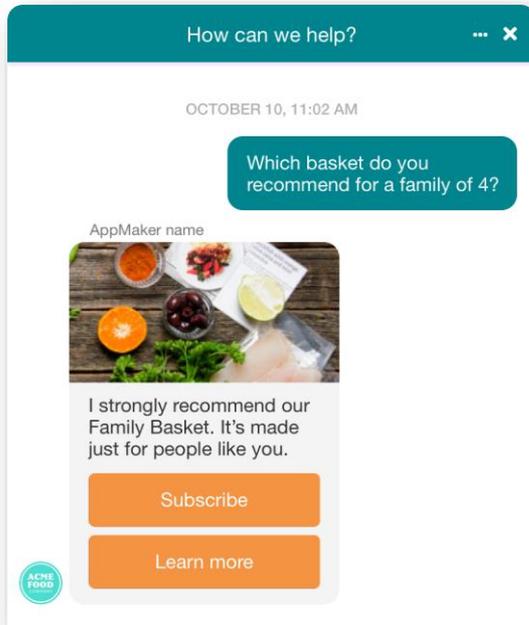
Once a user taps the request button, Web Messenger will first ask for location permission and then send the user's location.



Compound Message

You can compose messages with multiple actions.

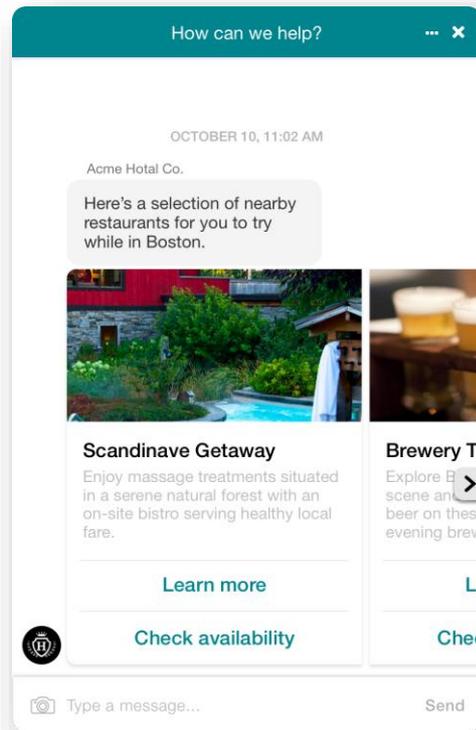
Compound message allows to send text, image and multiple buttons all in a single message.



Carousel

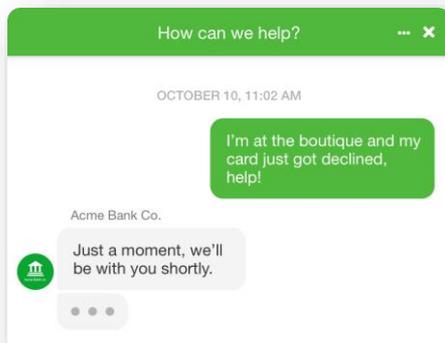
Send a horizontally scrollable set of cards that can contain text, image, and action buttons.

Carousels support up to a maximum of 10 message items. Each message item must include a title and at least one supported action.



Typing Status

Display a typing indicator. Web Messenger can show that the agent is typing with a typing animation. The API allows to specify an agent name and avatar.



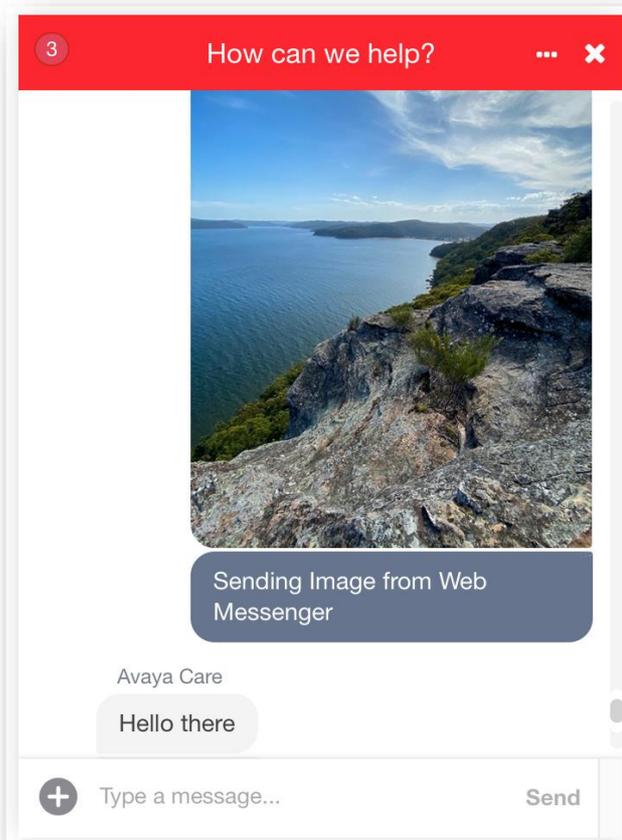
Badge for unread messages

When the user receives new messages, a badge containing the count of unread messages is displayed in the header of the Web Messenger if the user's view is not at the bottom of the conversation.

The badge is removed when the messages are marked as read with the following user actions:

- Scroll to bottom of conversation
- Focus or type in the chat input
- Click on the Web Messenger header or badge
- Click on the Web Messenger widget
- Send a message from a linked channel

Note that unless the user sends a message from a linked channel, no other user actions may remove the badge while the user is viewing the settings.



Getting Started

Avaya DevConnect offers a bundle Avaya In-app messaging SDK. This bundle includes SDK for each platform (iOS, android), API reference docs and a basic implementation of a mobile messaging app and Web Messenger that uses the SDK to send messages from an android Device to your Avaya contact center Solution.

To get started, download the bundle from Avaya DevConnect <http://www.avaya.com/devconnect>



What's included in the bundle?

```
— web
  |— distribution
  |   |— files
  |   |   |— cdnPackage
  |   |   |— docs
  |   |       |— README.md
  |   |       |— HOWTODEPLOY.md
  |   |— README.md
  |   |— configure
  |— Avaya\ SDK\ EULA.pdf
```

./distribution: contains a cdnPackage you can configure for your environment and host.

./distribution/HOWTODEPLOY.md: contains instructions on how to use configure script to generate cdnPackage for your environment.

./distribution/files/docs/README.md: Instructions on how to integrate the library on a website will be available in the 'README.md' file inside the generated folder.

The bundle includes everything you need to build and host the library for your environment (CDN) and further documentation to integrate Web Messenger to your website.

API reference

```
README.md

AvMessagingSdk Web Messenger

The AvMessagingSdk Web Messenger will add live web messaging to your website or web app.

Usage

Script Tag

Add the following code towards the end of the head section on your page and replace <integration-id> with your integration id at the end of the script.

<script>
  !function(e,n,t,r){function s(){try{var e;if(("string"==typeof this.response?
</script>

then initialize the Web Messenger by placing this snippet towards the end of the body section of your page.

<script>
  AvMessagingSdk.init({integrationId: '<integration-id>'}).then(function() {
    // Your code after init is complete
  });
</script>
```

README.md this will help you discover all the setting and options along with sample code for features described in the guide.

Location:

[././bundle/web/distribution/README.md](http://./bundle/web/distribution/README.md)

Browser support

Web Messenger supports all popular browsers.

Desktop

- Chrome: Latest and one major version behind
- Edge: Latest and one major version behind
- Firefox: Latest and one major version behind
- Internet Explorer: 11+
- Safari: Latest and one major version behind

Mobile

- Stock browser on Android 4.1+
- Safari on iOS 8+

The Web Messenger is optimized to be rendered within a viewport for mobile devices. User experience may be degraded without a viewport.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Other browsers

Web Messenger is likely compatible with other and older browsers but only the above versions have been tested and are supported.

Installation

The easiest way you can include the Web Messenger on your web page is using the Script Tag method

Step 1: Include the Web Messenger on your web page

Add the sample code snippet (from [API Reference](#)) towards the end of the `head` section on your HTML page and replace `<integration-id>` with your integration ID.

You can contact Avaya DevOps Team to get the integration ID for Web Messenger Integration to your **Avaya IX Digital Connection** app.

Step 2: Initialize with your new app ID

Once the script has been included on your web page, you're almost done. Simply initialize the Web Messenger using the code snippet in the [API Reference](#)

Keeping up to date

Web Messenger is updated periodically, you will need to update to the new version available via Avaya DevConnect. Each version is posted with corresponding release notes.

Region configuration

Web messenger is supported in the following regions:

Name	Region identifier
United States	Leave unspecified
European Union	eu-1

To target the EU region for example the region identifier is passed to `AvMessagingSdk.init()`:

Refer to [API Reference](#) docs for more details and sample code

You can contact your Avaya DevOps Team to verify region setting for your **Avaya IX Digital Connection** app.

Login with userId and JWT

After the Web Messenger is initialized, your user can start sending messages right away. These messages will show up on the business side under a new `appUser`. However, these `appUsers` will not yet be associated to any user record you might have in an existing user directory.

If your application has a login flow, or if a user needs to access the same conversation from multiple devices, this is where the `login` method comes into play. You can associate users with your own user directory by assigning them a `userId`. You will then issue each user a `jwt` credential during the login flow. You can read more about this in the [Authenticating users](#) section.

Authentication delegate

The Web Messenger offers an `onInvalidAuth` which will be called when an invalid authentication token has been sent to IX Digital Connection. It allows you to provide a new token for all subsequent requests. The request that originally failed will be retried up to five times.

Refer to [API Reference](#) docs for more details and sample code

See [Expiring JWTs on SDKs](#) for more information.

Customization

The Web Messenger includes a set of built-in customization options which you can leverage in two ways.

You can contact Avaya DevOps Team to verify settings for your **Avaya IX Digital Connection** app or you can do the settings using `AvMessagingSdk.init`

Please note that the settings included in the `AvMessagingSdk.init` method override the settings set remotely. Changes to Web Messenger settings may take up to 15 seconds to take effect.

Embedded mode

Supported by the `AvMessagingSdk.init` method.

To embed the widget in your existing markup, you need to pass `embedded: true` when calling `AvMessagingSdk.init`. By doing so, you are disabling the auto-rendering mechanism and you will need to call `AvMessagingSdk.render` manually. This method accepts a DOM element which will be used as the container where the widget will be rendered.

Refer to [API Reference](#) docs for more details and sample code

The embedded widget will take full width and height of the container. You must give it a height, otherwise, the widget will collapse.

Strings customization

Supported by the `AvMessagingSdk.init` method.

The Web Messenger lets you customize any strings it displays by overwriting its keys. Simply add the `customText` key in your `AvMessagingSdk.init` call and specify new values for the keys used in. You can find all available keys in [API Reference](#). If some text is between `{}`, or if there is an html tag such as `<a>`, it needs to stay in your customized text.

For example:

```

AvMessagingSdk.init({
  integrationId: '<integration-id>',
  customText: {
    headerText: 'How can we help?',
    inputPlaceholder: 'Type a message...',
    sendButtonText: 'Send'
  }
}).then( ....);

```

Date localization

Supported by the `AvMessagingSdk.init` method.

When using strings customization to translate the interface, you will also want to have Web Messenger show the date and time in the right language. To do this, simply pass `locale` at initialization time. You might also want to override the timestamp format to match your language. You can learn more about formats in [API Reference](#).

```

AvMessagingSdk.init({
  integrationId: '<integration-id>',
  locale: 'fr-CA'
  customText: {
    // ...
    conversationTimestampHeaderFormat: 'Do MMMM YYYY, hh:mm',
    // ...
  }
}).then( ....);

```

The `locale` options is using the `language-COUNTRY` format. Language codes can be found [here](#) and country codes [here](#). The country part is optional, and if a country is either not recognized or supported, it will fallback to using the generic language. If the language isn't supported, it will fallback to `en-US`. A list of supported locale can be found on [date-fns Github repository](#).

The `locale` option only affects date and time localization. It doesn't translate the interface as Web Messenger doesn't provide built-in translations. Translating the interface can be achieved through [Strings customization](#).

Display Style

Supported by the `AvMessagingSdk.init` method or *Avaya DevOps Support*.

The Web Messenger can be displayed as a button or as a tab. The default style is the `button` mode.

When the display style is a button, you have the option of selecting your own button icon. The image must be at least 200 x 200 pixels and must be in either JPG, PNG, or GIF format.

Button Size

You can customize the size of the button by setting a `buttonWidth` and `buttonHeight`. When not provided, the button will have a default size of 58 x 58 pixels.

AvMessagingSdk.init

Specify the `displayStyle`, `buttonIconUrl`, `buttonWidth` and `buttonHeight` in the call to `AvMessagingSdk.init`

```
AvMessagingSdk.init({
  integrationId: '<integration-id>',
  // ...
  displayStyle: 'button',
  buttonIconUrl: 'https://myimage.png',
  buttonWidth: '90',
  buttonHeight: '90'
  // ...
}).then( ....);
```

Fixed Intro

Supported by the `AvMessagingSdk.init` method or *Avaya DevOps Support*.

You can set the introduction pane to fixed mode by setting `fixedIntroPane` to `true`. When set, the pane will be pinned at the top of the conversation instead of scrolling with it. The default value is `false`.

Refer to [API Reference](#) docs for more details and sample code

Colors

Supported by the `AvMessagingSdk.init` method or *Avaya DevOps Support*.

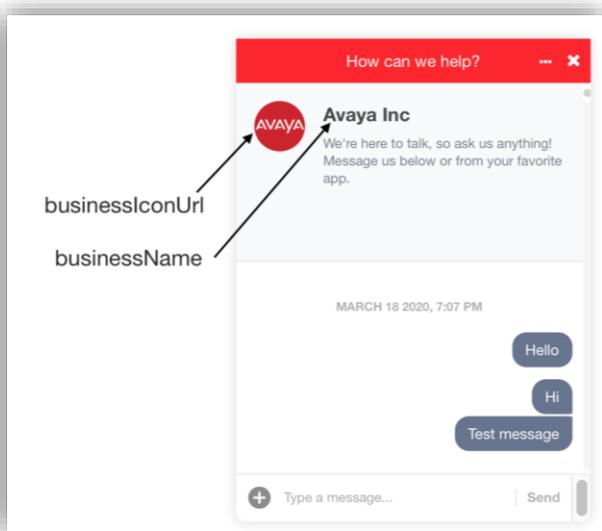
The supported color customizations are:

- The **Brand Color** customizes the color of the messenger header. It is also used for the color of the button or tab in idle state, as well as the color of the default app icon. If no color is specified, the brand color will default to `#65758e`.
- The **Conversation Color** customizes the color of customer messages and actions in the footer. If no color is specified, the conversation color will default to `#0099ff`.
- The **Action Color** changes the appearance of links and buttons in your messages. It is also used for the 'Send' button when it is in active state. If no color is specified, the action color will default to `#0099ff`.

Refer to [API Reference](#) docs for more details and sample code

Business profile

Supported by the `AvMessagingSdk.init` method or *Avaya DevOps Support*.



You can customize your business branding with the `businessName` and `businessIconUrl` settings.

For the `businessIconUrl` setting, the image must be at least 200 x 200 pixels and must be in either JPG, PNG, or GIF format.

If the branding settings are not set, they fall back to the app's settings. The app name is used as the `businessName`, and the app icon is used as the `businessIconUrl`.

Refer to [API Reference](#) docs for more details and sample code

Background Image

Supported by the `AvMessagingSdk.init` method or *Avaya DevOps Support*.

You can customize the background image in your SDK with the `backgroundImageUrl` setting. The image will be displayed at its full size, and tiled if it is not large enough to fill the conversation.

Refer to [API Reference](#) docs for more details and sample code

Sound notification

Supported by the `AvMessagingSdk.init` method or *Avaya DevOps Support*.

By default, a sound notification will be played when a new message comes in and the window is not in focus.

To disable this feature, you need to add the `soundNotificationEnabled` option to the `AvMessagingSdk.init` call, like this:

```
AvMessagingSdk.init({
  integrationId: '<integration-id>',
  // ...
  soundNotificationEnabled: false // Add this line to your init call
  // ...
}).then( .... );
```

Browser storage

Supported by the `AvMessagingSdk.init` method.

By default, the Web Messenger will store the identity of anonymous users in the `localStorage` of the browser.

Using the `localStorage` will persist the user identity throughout browser sessions (including page reloads and browser restarts). To clear the user identity once the browser is closed, use `sessionStorage` instead. [Learn more](#)

```
AvMessagingSdk.init({
  integrationId: '<integration-id>',
  browserStorage: 'sessionStorage' // Add this line to your init call
  // ...
}).then( .... );
```

Menu items

Supported by the `AvMessagingSdk.init` method.

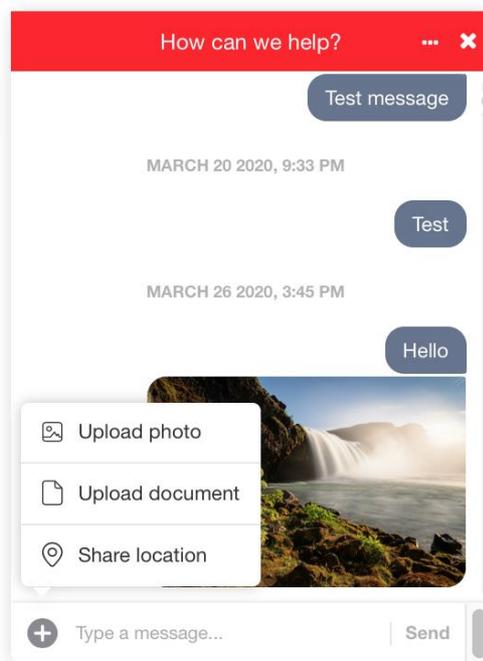
The Web Messenger features a menu that allows the user to send various message types. The types displayed in this menu can be customized, or the menu can be hidden altogether.

If you want to control this menu, add the `menuitems` option to the `AvMessagingSdk.init` call:

```
AvMessagingSdk.init({
  integrationId: '<integration-id>',
  menuitems: {
    imageUpload: true,
    fileUpload: true,
    shareLocation: true
  },
  // ...
}).then( .... );
```

To hide the menu completely, override the `menuitems` option as follows:

```
AvMessagingSdk.init({
  integrationId: '<integration-id>',
  menuitems: {},
  // ...
}).then( .... );
```



Filtering and transforming messages

The Web Messenger allows you to filter and transform messages by using the `beforeDisplay` delegate. To set a delegate, use the `setDelegate` method.

This may be useful if you want to conditionally filter specific messages or transform their structure. See the [Message Schema](#) section to learn more on the message properties that you can access and modify. You may also want to set your own properties using the `beforeSend` delegate.

Refer to [API Reference](#) docs for more details and sample code

Filtering messages

The following example sets a `beforeDisplay` delegate method to check if a message has been marked as archived in the metadata for a specific conversation (this can be done using the `beforeSend` delegate method as well). If the field `isArchived` is true, we instruct the SDK to hide the message by returning `null`. If not, we simply return the original message unmodified.

```
const delegate = {
  beforeDisplay(message, data) {
    if (data.conversation._id === '<conversation-id>' && message.metadata &&
        message.metadata.isArchived) {
      return null;
    }
    return message;
  }
};

AvMessagingSdk.init({
  integrationId: '<integration-id>',
  delegate
});
```

Transforming business messages

The following example sets a default author name on messages for a specific conversation that lacks one if they are originating from the business. We first check if the message is sent by the `appMaker` and if a name has been specified. If not, we set it and return the modified message.

```
const delegate = {
  beforeDisplay(message, data) {
    if (data.conversation._id === '<conversation-id>' && message.role === 'appMaker'
        && !message.name) {
      message.name = 'Acme Bank';
    }
    return message;
  }
};

AvMessagingSdk.init({
  integrationId: '<integration-id>',
  delegate
});
```

Transforming user messages

The following example adds the URL of the page the user is visiting while messaging a business for a specific conversation. This can be useful when users navigate within a site while messaging. We first check if the message role is `appUser` and then add a `currentUrl` attribute in the `metadata` property.

```
const delegate = {
  beforeSend(message, data) {
    if (data.conversation._id === '<conversation-id>' && message.role === 'appUser') {
      message.metadata = {
        ...message.metadata,
        currentUrl: window.location.href
      };
    }
    return message;
  }
};

AvMessagingSdk.init({
  integrationId: '<integration-id>',
  delegate
});
```

Transforming postback

The following example adds the URL of the page the user is visiting while actioning a postback for a specific conversation. This can be useful to tailor the reaction to the postback in accordance to the current page. We add a `currentUrl` attribute in the `metadata` property.

```
const delegate = {
  beforePostbackSend(postback, data) {
    if (data.conversation._id === '<conversation-id>') {
      postback.metadata = {
        ...postback.metadata,
        currentUrl: window.location.href
      };
    }
    return postback;
  }
};

AvMessagingSdk.init({
  integrationId: '<integration-id>',
  delegate
});
```

Whitelisting Domains

By default, the Web Messenger can be initialized from any domain with the use of the `integrationId`. To limit domains where Web messenger can be used, you can use the `originWhitelist` setting.

When the `originWhitelist` is set, It restrict the access to only the origins listed in the array by implementing [Cross Origin Resource Sharing \(CORS\)](#). More specifically, it will look at the `Origin` header for every HTTP request and block it if the `Origin` is not part of the `originWhitelist` array.

You can contact your Avaya DevOps Team for *originWhitelist* setting request

An [origin](#) is defined as the combination of the protocol, hostname and, optionally, a port (e.g. `https://avaya.com`).

Notes:

- `localhost` is supported as a valid hostname in the origin, so you can test when doing local development.
- Wildcard subdomains are not currently supported, so you need to list all of your supported domains individually.

Content Security Policy

If your deployment requires [CSP compatibility](#), add the following meta tag to your configuration.

```
<meta
  http-equiv="Content-Security-Policy"
  content="
    connect-src
      wss://*.{YOUR_DOMAIN_NAME}
      https://*.{YOUR_DOMAIN_NAME}};
    font-src
      https://*.{YOUR_DOMAIN_NAME}};
    script-src
      https://*.{YOUR_DOMAIN_NAME}};
    style-src
      https://*.{YOUR_DOMAIN_NAME}};
    img-src
      blob:
      https://*.{YOUR_DOMAIN_NAME}};"
/>
```

Note that an equivalent configuration can be done [server side](#).

Note that your CSP configuration should also include any domains used to host images or files sent in messages. If you require blob: to be excluded for img-src, you must disable the image upload feature via the [init settings](#) function.

Refer to [API Reference](#) docs for more details and sample code

Appendix A – Users Authentication

If your software or application has an existing user authentication method then you can optionally federate those identities with **Avaya IX Digital Connection** by issuing a [JSON web token](#) (JWT). A JWT is required to protect the identity and data of these users. This option requires your app to be connected to your own secure web service. There are JWT libraries available supporting a wide variety of popular languages and platforms.

First, you must assign a `userId` to each of your users. The `userId` will uniquely identify your users within Avaya IX Digital Connection and the JWTs you issue serve as a signed proof that your software or app has successfully authenticated that user.

A `userId` is a string that can have any value you like, but must be unique within a given **Avaya IX Digital Connection** app. Examples of `userId`s include usernames, GUIDs, or any existing ID from your own user directory. The `userId` should map to a unique identity in your existing user directory. The `userId` should always reference an external entity; in other words you should not reuse any id that was assigned by **Avaya IX Digital Connection** as a `userId`. When choosing a `userId` you should also ideally avoid using user properties that change, like a phone number.

To log in with a JWT:

1. Generate an API key for your **Avaya IX Digital Connection** app.

You can contact your Avaya DevOps Team to generate *key ID* and *secret* for your **Avaya IX Digital Connection** app.

2. Implement server side code to sign new JWTs using the **key ID** and **secret provided**. The JWT header must specify the key ID (`kid`). The JWT payload must include a `scope` claim of `appUser` and a `userId` claim which you've assigned to the app user. Make sure the `userId` field is formatted as a String. If you use numeric ids, the `userId` must be a String representation of the number - using a number directly will result in an invalid auth error.
3. Issue a JWT for each user. You should tie-in the generation and delivery of this JWT with any existing user login process used by your app.
4. Initialize Avaya In-App Messaging SDK in your website or app. See instructions for [Web Messenger](#) in this document.

A node.js sample is provided below using `jsonwebtoken >= 6.0.0`

```

var jwt = require('jsonwebtoken');
var KEY_ID = 'app_5deaa3531c7f940010cc4ba4';
var SECRET = 'BFJJ88naxc5PZNAMEU9KpBNTR';

var signJwt = function(userId) {
  return jwt.sign(
    {
      scope: 'appUser',
      userId: userId
    },
    SECRET,
    {
      header: {
        alg: 'HS256',
        typ: 'JWT',
        kid: KEY_ID
      }
    }
  );
};

```

5. Call `AvMessagingSdk.login` with your `userId` and `jwt`:

Web (JavaScript):

```

AvMessagingSdk.login('user-id', 'jwt').then(
  function() {
    // Your code after login is complete
  },
  function(err) {
    // Something went wrong during login. Your JWT might be invalid
  }
);

```

If your API key is ever compromised you can generate a new one. Avaya IX Digital Connection will accept a JWT as long as it contains all required fields and is signed with any of your Avaya IX Digital Connection Conversations app's valid API keys. Deleting an API key will invalidate all JWTs that were signed with it.

Expiring JWTs on SDKs

If you desire to generate credentials that expire after a certain amount of time, using JWTs is a good way to achieve this.

The exp (expiration time) property of a JWT payload is honoured by the IX Digital Connection API. A request made with a JWT which has an exp that is in the past will be rejected.

Keep in mind that using JWTs with exp means that you will need to implement regeneration of JWTs, which demands additional logic (Android, iOS or Web Messenger) in your software.

JWTs are required to identify your users by a custom identifier (userId) in SDKs. In this case, JWTs are signed with an app API key with a scope of appUser, and an additional payload property userId.

Sample JWT Structure with expiry

Header:

```
{
  "alg": "HS256",
  "typ": "JWT",
  "kid": "<app-key-id>"
}
```

Payload:

```
{
  "scope": "appUser",
  "exp": "1542499200",
  "userId": "<user-id>"
}
```

Note: expiry field - timestamp representing 2018-11-18T00:00:00+00:00

Users on multiple clients

You may have a single user logging in as the same `userId` from multiple clients. For example, they have your app installed on both their iPhone and their iPad or multiple android devices. You might also have Avaya IX Digital Connection integrated in both your mobile app as well as on your web site.

Once a user has been logged in to Avaya IX Digital Connection, they will see the same conversation across each of these clients.

Omitting the userId

Avaya IX Digital Connection will work perfectly fine without a `userId`. Profile information can still be included, and the user can take advantage of all rich messaging features, but the user will only be able to access the conversation from the client they're currently using. Without a `userId`, if the same individual opens Avaya IX Digital Connection on a new client, or runs your web app in an incognito browser session, they will see a newly created empty conversation when they open Avaya IX Digital Connection, and on the contact center

(business) side they will be represented as two distinct `appUsers`. This will happen even if you specify the same profile information in both cases.

A `userId` can also be omitted at first and added at a later time. If you deploy an update to your app that assigns an existing user with a new `userId` that they didn't have before, any existing conversation history they have will be preserved and their messages will start being synchronized across all clients where that `userId` is being used. This is particularly useful if a user opens Avaya IX Digital Connection and starts a conversation before having logged in to your app or website.

Switching users

If your app allows a shared client to switch between multiple user identities you can call the `login` API multiple times to switch between different `userIds`.

Logging out

Your app may have a logout function which brings users back to a login screen. In this case you would want to revert IX Digital Connection to a pre-login state. You can do this by calling the `logout` API.

Calling `logout` will disconnect your user from any `userId` they were previously logged in with and it will remove any conversation history stored on the client. Logging out will *not* disable Avaya IX Digital Connection. While logged out, the user is free to start a new conversation but they will show up as a different `appUserId` on the business end.

Web (JavaScript):

```
AvMessagingSdk.logout().then(  
  function() {  
    // Your code after logout is complete  
  },  
  function(err) {  
    // Something went wrong during logout  
  }  
);
```

Appendix B - Managing Users

In addition to the information automatically collected and stored for each of a user's clients, an **appUser** itself can have metadata and profile information attached to it, in order to better understand the context and the history of the user.

The appUser

In the IX Digital Connection lexicon, user (usually referred to as an app user or appUser) refers to an end-user of your platform or a customer of your business. The following are all examples of what IX Digital Connection refers to as an **appUser**:

- A visitor to your Website
- The holder of an SMS number
- A user of your mobile app
- A member of the public on Facebook Messenger

Profile information can be added at runtime with the mobile and web SDKs. There are two types of profile information fields: *structured* and *unstructured*.

Structured Fields

Structured fields are properties that IX Digital Connection has identified as common across many use cases, and has exposed as common properties across all users, when present. The currently supported structured fields are:

- **givenName**, also referred to as **firstName** in some contexts, which represents the user's given name
- **surname**, also referred to as **lastName** in some contexts, which represents the user's surname
- **signedUpAt**, which is the date when the user first started using your service, or when they first became a customer. If not customized, this field is automatically populated to be the date the user was created in , which is most likely the moment when the user messaged you for the first time.
- **email**, which represents the user's email address.

Unstructured Fields

Unstructured fields, also referred to as "custom properties", are a set of key/value pairs that are specific to your application domain, or use case. These fields are stored under the **properties** field of an appUser, and can have values of type **Number**, **String**, or **Boolean**.

Custom properties are limited to 4KB per users. Each custom property 'key' is limited to 100B and each custom property 'value' is limited to 800B. Exceeding characters will be truncated. An error will be returned if an appUser is in the process of being created, or updated, and the sum of all custom properties' sizes is over the 4KB limit.

Adding properties using the SDKs

Each of IX Digital Connections' web and mobile SDKs support attaching properties to a user at runtime. The details of when and how these properties are uploaded to the server is handled automatically by the SDKs, so in general you should not need to worry about this detail. However, the process is documented here for completeness.

On Android and iOS, when a user property is set using one of the SDK methods, the properties are immediately serialized to disk until they can be uploaded to the server. Changes to user properties are uploaded in batches at regular intervals while the app is in the foreground, as well as just before the app is sent to the background, or immediately before a message is sent by the user. If the application exits unexpectedly, or the user has intermittent internet connection or no internet connection, the properties will remain on disk until the upload eventually succeeds (even across app launches). If the user does not yet exist (i.e. the user has not yet sent their first message, and the [startConversation](#) method has not been called), the properties are still tracked and stored on disk until the user is eventually created, and they will be uploaded as part of the user creation flow.

On Web, the user properties are stored in memory, and uploaded in batches at regular intervals, or immediately before a message is sent by the user. If the user does not yet exist (i.e. the user has not yet sent their first message, and the [startConversation](#) method has not been called), the properties are still tracked and stored in memory until the user is eventually created, and they will be uploaded as part of the user creation flow. In contrast to the Android and iOS SDKs, the Web Messenger does not store any user property information on disk - if the browser window is closed before the user is created, then the properties will be discarded.

Web Messenger (JavaScript):

```
AvMessagingSdk.updateUser({
  givenName: 'Steve',
  surname: 'Brule',
  email: 'steveb@channel5.com',
  signedUpAt: Date.now(),
  properties: {
    premiumUser: true,
    numberOfPurchases: 20,
    itemsInCart: 3,
    couponCode: 'PREM_USR'
  }
});
```

The `addProperties` method accepts a `Map` containing the properties to add. This dictionary must have keys that are type `String` and values that are either `String`, `Integer`, `Long`, `Float`, `Double`, `Boolean`, or `Date`. If your map contains any other data type as a value, then `toString` will be called on the object and the resulting `String` will be added as a property.

Appendix C - Web Messenger FAQ

How can I automatically open the Web Messenger on my website?	You can use the function <code>open()</code> function of the Web Messenger. Simply call this function after you initialize on your web page.
How can I customize the texts that are displayed on the Web Messenger?	Customizing labels is easy, see string customization for more details. You can find the list of all customizable strings in the api docs .
How can I customize the Web Messenger?	You have two ways for changing the appearance of the widget: <ul style="list-style-type: none">• Use our built in style selector in the dashboard.• Fork the Web Messenger and change whatever aspects you like.
How can I reset an anonymous conversation on the web (for testing)?	On the dev console, navigate to <code>Application tab > Local storage > your website</code> and clear the entries.
How can I embed the Web Messenger in a container of my website?	You can embed the Web Messenger by using this function .
The Web Messenger doesn't display well on mobile. How can I fix that?	Make sure you have this code in your HTML file: <pre><meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1"></pre>
How can I update the user data?	You can update any information that you want on the user by using the <code>updateUser()</code> method. described here .
Why is my profile picture not updating?	Did you enable "Use app icon and a single team name for all messages" on the Avaya IX Digital Connection settings?

	<p>If you did, the image may be persisted in your cache. Refresh it and try again.</p> <p>Or contact Avaya DevOps Team.</p>
Nothing shows up when I add script?	<p>There's a couple of quick checks that you can do if our script doesn't load:</p> <ol style="list-style-type: none"> 1. Make sure you separate the 2 scripts we provide 2. Make sure your integration ID doesn't contain "copy" <p>If that doesn't do the trick, note that there are a couple of libraries we conflict with currently: mootools, requirejs & Prototype js. Make sure that you're not using one of them.</p>
How can I insert your widget in a container?	<p>You can embed anywhere on your webpage now by following the instructions of the docs.</p>
How do I disable the sound notifications?	<p>You can follow these instructions to disable the sound notifications.</p>
Can I put the Web Messenger on two different websites, and also change the header?	<p>Yes, you can place the web chat on as many sites as you like. To change the header, just set a different <code>introductionText</code> property in your <code>AvMessagingSdk.init</code> call on each site.</p>

Appendix D – certificate configuration for file transfer feature

You need to generate client cert using **openssl**. To do it you need to run the following commands (in order). In addition, for testing it is better to have the same password (6 and more characters):

```
openssl genrsa -aes256 -out client.key 2048
```

```
openssl pkey -in client.key -out client_test_fixed.key
```

(after this command you can delete **client.key**. **client_test_fixed.key** which was generated after this command should be left and renamed to **client.key**)

```
openssl req -x509 -sha256 -new -key client.key -out client.csr
```

(during command execution you need to answer the following questions)

Country Name (2 letter code) [AU]:

State or Province Name (full name) [Some-State]:

Locality Name (eg, city) []:

Organization Name (eg, company) [Internet Widgits Pty Ltd]:

Organizational Unit Name (eg, section) []:

Common Name (e.g. server FQDN or YOUR name) []:

Email Address []:

```
openssl x509 -sha256 -days 3652 -in client.csr -signkey client.key -out client.crt
```