Alchemer Web Integration Guide

This document will show you how to integrate Alchemer Web into your website, configure it, and test to make sure it's working properly.

> If you have additional implementation questions after reading this implementation guide, the Alchemer Web FAQ page may be able to help.

Supported Platforms

We use a sliding window in our browser support policy for Alchemer Web with the most common browsers in mind:

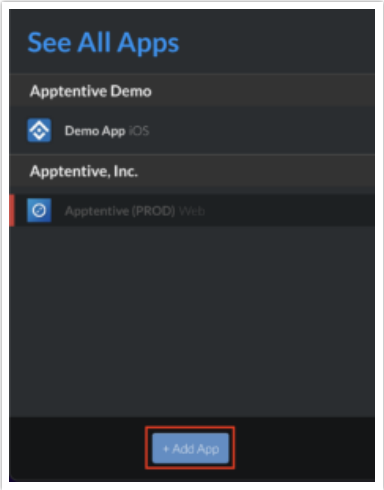
- Chrome: last 3 versions
- Firefox: last 3 versions
- Safari: last 3 versions
- Edge: last 3 versions

Registering Applications

To get started, an "application" will need to be created in your Alchemer Mobile account for each of the entities you wish to use Alchemer on. An application can be used to track a group of websites or just a single one and can be created on the dashboard.

> *Note: In order to create a web application, your Alchemer account will need the web entitlement. If this has not been enabled, please contact your CSM to get that added.*

Once logged in, hover over the icon in the top left hand corner and then click on the "Add App" button at the bottom of the slideover.



Once on the “Add App” screen, select “Web” for the platform, fill out the rest of the form, and click the create button to finish up.

> *Note: Make sure to use a different application for each website unless wanting to track them as a single entity.*

Adding Alchemer to Your Website

Once an application has been created, integrating Alchemer Web to your website involves adding a snippet of code just before the end of the `<head>` tag. To get the correct snippet to use, if the desired application is currently selected from the dashboard you can use this link.

Otherwise, follow these steps:

- Login to the dashboard
- Select the intended web application registered previously using the icon in the top left corner
- Navigate to the settings tab
- Click on the “API & Development” section in the left sidebar
- Look for the “Website Snippet” area

Once there, copy and paste the full contents of the text area into the website(s) that will be tracked under this application. This snippet will need to be loaded on every page in order for Alchemer Web to function.

The purpose of this is two-fold — it first provides a minimal bootstrap so that the SDK function methods are available on page start and second it downloads the full implementation to be used for the rest of the session. This allows for code to be written that invokes the SDK prior to the script being fully loaded on the page. Any call that is made to those methods before the full initialization is complete will be queued and sent off in calling order once ready.

Mobile Websites

For websites that are responsive or intended to be viewed on a mobile device, it is important that the site has the proper viewport scale set in the `<meta/>` tags in the `<head>` section. If a viewport is not already set, we would recommend adding this line to ensure an ideal viewing experience:

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

Integration

Now that an application is created and the snippet is integrated into the site, it’s time to create a conversation and begin sending events!

Creating a Conversation

> Make sure the conversation is created before an event engages. If the event is called before a conversation is created your surveys may surface inconsistently.

To begin sending data to Alchemer, the first thing that needs to be done is creating a conversation. A conversation is the primary way to interact with customers, and what all events and other data is correlated with. You can think of a conversation like an on-going dialog with people engaging with your site and using it to track usage, engagement, and other data points throughout an application that can be used to trigger interaction points.

To start a conversation, there is a method on the SDK named `createConversation` that needs to be called. This method can simply be invoked or called with additional starting data such as `person` details, `app_release` information, or custom `device` data. All of this information can be

added or updated after the conversation has been created initially, but it is recommended to add it at conversation creation if possible. Below are examples of creating a conversation in both scenarios:

```
// Create a conversation with no additional metadata
// This will create an anonymous session
ApptentiveSDK.createConversation();
```

```
// Or pre-populate conversation details like person or device data
ApptentiveSDK.createConversation({
  app_release: {
    version: '1.0.0' // <= Your website's version
  },
  person: {
    unique_token: 'UNIQUE_IDENTIFIER', // <= A unique identifier meaningful to you (such as account id)
    name: 'Full Name',
    email: 'user@domain.tld',
    custom_data: {
      age: 30,
      premium: true
    }
  },
  device: {
    custom_data: {
      flash: true,
    }
  }
});
```

Note: For the `unique_token` property, it's strongly recommended to use an id in UUID/GUID format. These are long, random, and well distributed id's that contain no sensitive information.

Add Customer Information

If a customer's details aren't available at the start of a conversation, it can be added at any time using the `identifyPerson` method. When calling this method, there **must be a unique identifier** used for the person in order to identify them properly and be able to connect them to previous data in the system. We recommend using an identifier that will not change over the lifetime of the integration such as an account id or user id.

```
ApptentiveSDK.identifyPerson({
  unique_token: 'UNIQUE_IDENTIFIER',
  name: 'Full Name',
  email: 'user@domain.tld'
});
```

Sending Events

Refer to the Alchemer Web FAQ article for more information on creating events and to see a

few examples.

If a conversation is the heart of an Alchemer Web integration, events are heartbeats that continuously inform and strengthen that dialog. Events are both records of an action within a website being performed as well as an opportunity to show an Interaction to a customer. When building out this integration, we recommend thinking about places where significant actions occur (e.g., login) and places where it would be desirable to interact with a customer (e.g., checkout).

Remember, the more events added during integration the more you will learn about your customers and the more touch points you will have available with them in the future. To emit an event, use the `engage` method of the SDK like this:

```
// Add an event by capturing an "engagement" by the customer
ApptentiveSDK.engage('eventName')

// This could be a system level event like a login
ApptentiveSDK.engage('login');

// Or a generic data point like a page view
ApptentiveSDK.engage('pageView-settings');

// All the way to a specific button click
ApptentiveSDK.engage('cart-button');
```

Note: Each Event may be used in multiple places throughout your code. Events are grouped by their name, so for any differentiation between sections or contexts you will need to use a different Event name.

Message Center

With the Alchemer Message Center for Web, your customers can send feedback and you can easily send replies directly to their emails. Message Center can be launched in two ways — from a static button on your website or from another Alchemer Mobile interaction (such as the Love Dialog).

To launch it directly from a button, call the following method:

```
ApptentiveSDK.showMessageCenter()
```

To launch it from another interaction (specifically from the Love Dialog when a customer says “No”) please configure this through the dashboard to connect message center to that action.

We recommend using a combination of both methods. This is the best of both worlds and allows customers to contact you when they need help, as well as prompting them for specific feedback at meaningful moments.

Replying to Customers

In order to reply to customers, you must have their email address. You can make this a requested

or required field from your Alchemer Mobile Dashboard and a form will automatically be rendered when message center opens from any source. If a name and email has been identified through the above methods, these fields will not display since that information is already available.

Configure Interactions

Once you have added several Events to your app, you can start adding Interactions through your Alchemer dashboard. See this documentation for more information about configuring interactions.

Interaction Styles

You have some control over the appearance of the Interactions on your site through the Interactions -> Interaction Styling section of your Alchemer Mobile account. Any style modifications outside of this are not recommended and not supported.

Custom Data

You can send custom data associated with either the device, or the person using the app. This is useful for sending user IDs and other information that helps you support your users better. Custom Data can also be used for configuring when interactions will run.

```
ApptentiveSDK.buildDevice({
  custom_data: {
    flash: true,
    html: false
  }
});
```

```
ApptentiveSDK.identifyPerson({
  unique_token: 'UNIQUE_IDENTIFIER',
  custom_data: {
    purchase: true,
    age: 30
  }
});
```

Changing Options

In some instances you may want to change the default settings in the SDK. To do so you can use the `setOption` method of the SDK.

Option: domNode

In some environments you may want to render Interactions into a specific DOM node other than `body` for layering or presentation considerations. This can be set with the `domNode` option which takes any valid CSS selector string. This can and should be called as early as possible when setting up the SDK.

```
// This option can have any valid CSS selector string
// and will be used by document.querySelector(customOption)
ApptentiveSDK.setOption('domNode', '#target-dom-node')
```

Note: If a valid element is not able to be found, the SDK will fallback to using the `body` element.

Option: debug

When troubleshooting an integration it is often helpful to get logs out of the SDK to understand what is happening under the hood. To enable debug mode, use the `setOption` method to change the debug setting to `true`:

```
// This will enable logs in the browser console
ApptentiveSDK.setOption('debug', true)
```

Persisting Options

Any setting that is modified using this method will only persist on the current page instance. If you want these settings to persist (such as testing an integration across multiple page loads) you can instead set an object in localstorage that will be read in when the SDK initializes each page load.

```
// Persist SDK settings across pages
const sdkOptions = {
  debug: true,
  domNode: '#target-dom-node',
  readOnly: true, // <= Can only be used as a persistent setting
};

window.localStorage.setItem('ApptentiveSDKOptions', JSON.stringify(sdkOptions));
```

Related Articles