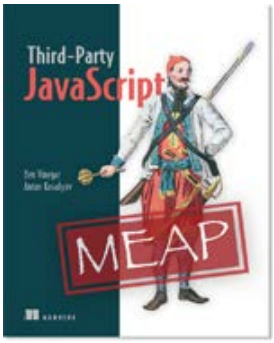
MANNING PUBLICATIONS



Third-Party JavaScript
By Benjamin A. Vinegar and Anton A. Kovalyov

In this green paper based on Third-Party JavaScript, the authors discuss some real-world use cases of third-party scripts: embedded widgets, analytics and metrics gathering, and client API wrappers.

Three Ways to Use Third-Party JavaScript

Third-party JavaScript code is code meant to be executed on someone else's web page. With access to that page's HTML elements and JavaScript context, third-party code can manipulate the page in a number of ways, which might include creating new elements on the Document Object Model (DOM), inserting custom stylesheets, or registering browser events for capturing user actions. For the most part, third-party scripts can perform any operation you might use JavaScript for on your own website or application but, instead, on someone else's.

In this article, we'll look at some real-world use cases of third-party scripts: embedded widgets, analytics and metrics gathering, and client API wrappers.

Embedded widgets

Embedded widgets (oftentimes called "third-party widgets") are perhaps the most common use case of third-party scripts. These are typically small, interactive applications that are rendered and made accessible on a publisher's website, but load and submit resources to and from a separate set of servers. Widgets can vary widely in complexity; they can be as simple as a graphic that displays the weather in your geographic location, or as complex as a full-featured instant messaging client.

Widgets enable website publishers to embed applications into their web pages with very little effort. They are typically easy to install; more often than not publishers need only insert a small HTML snippet into their web page source code to get started. Since they're entirely JavaScript based, widgets don't require the publisher to install and maintain any software that executes on their servers, which means less maintenance and upkeep.

Some businesses are built entirely on the development and distribution of embedded widgets. For example, there's Disqus, a web startup based out of San Francisco. Disqus develops a commenting widget (figure 1) that serves as a drop-in commenting section for blogs, online publications, and other websites. Its product is driven almost entirely by third-party JavaScript: It uses JavaScript to fetch commenting data from the server, render the comments as HTML on the page, and capture form data from other commenters—in other words, everything. The commenting widget is installed on websites using a simple HTML snippet that totals five lines of code.

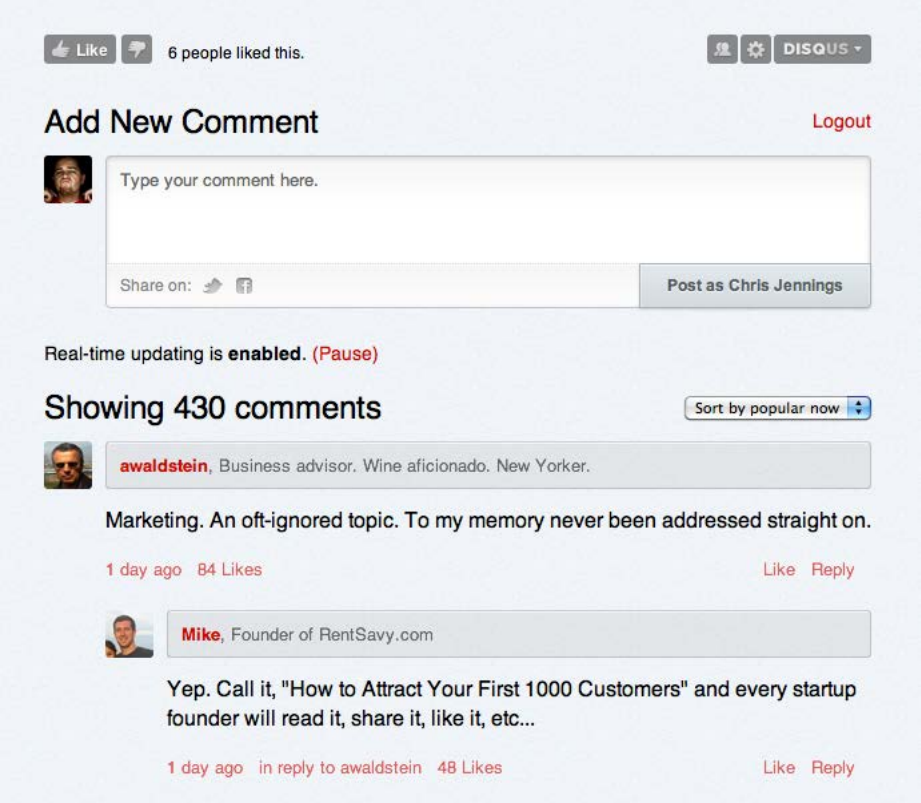


Figure 1 Readers of AVC.com congregate on the website's commenting section, powered by the Disqus commenting widget

Disqus is an example of a product that is only usable in its distributed form; you'll need to visit a publisher's page to use it. But widgets aren't always standalone products like this. Oftentimes, they're "portable" extensions of larger, more traditional stay-at-home web applications.

For example, consider Google Maps, arguably the web's most popular mapping application. Users browse to maps.google.com to view interactive maps of locations all over the world. Google Maps also provides directions by car and public transit, satellite imagery, and even street views using on-location photography.

Incredibly, all of this magic also comes in a widget flavor. Publishers can embed the maps application on their own web pages using some simple JavaScript code snippets obtained from the Google Maps website. On top of this, Google provides a set of public functions for publishers to modify the map contents.

Let's see how simple it is to embed an interactive map on our web page using Google Maps.

Listing 1 Initializing the Google Maps widget

```
<!DOCTYPE html>
<html>
  <head>
    <title>Google Maps Example</title>
    <script src="http://maps.google.com/maps?file=api&v=2"></script>

    <script>

    function initialize() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map_canvas"));
        map.setCenter(new GLatLng(37.4419, -122.1419), 13);
        map.setUIToDefault();
      }
    }

    </script>
```

```
  </head>
  <body onload="initialize()">
    <div id="map_canvas" style="width: 500px; height: 300px"></div>
  </body>
</html>
```

This code example begins by first pointing to the Maps JavaScript library using a simple script include. Then, when the body's onload handler fires, we check if the current browser is compatible, and if so, initialize a new map and center it at the given coordinates.[1] We're done, and all it took was roughly 10 lines of code—powerful stuff!

We looked at but two examples of embedded widgets. But really, any application idea is fair game for embedding on a publisher's page. In our own travels, we've come across a wide variety of widgets: content management widgets, widgets that play real-time video, widgets that let you chat in real time with a customer support person. If you can dream it, you can embed it.

Analytics and metrics

Third-party JavaScript isn't used exclusively in the creation of embedded widgets. There are other uses that don't necessarily involve graphical, interactive web page elements. Often, they're silent scripts processing information on the publisher's page without the user's ever knowing they're there. An example of this use case is in analytics and metrics gathering. One of JavaScript's most powerful features is that it enables developers to capture and respond to user events as they occur on a web page. From our third-party script, we can use event hooks to capture data about how the user interacts with the publisher's page and send the data back to our web service for storage and analysis. This might include tracking how long a visitor stays on a page before moving on, what content they saw while they were reading the page, and where they went afterwards. There are dozens of browser events your JavaScript code can hook into, from which you could derive hundreds of different insights.

Crazy Egg, another web startup, is one example of an organization that uses third-party scripts in this way. Their analytics product generates visualizations of user activity on your web page (see figure 2). To obtain this data, Crazy Egg distributes a script to publishers that captures the mouse and scroll events of web page visitors. This data is submitted back to Crazy Egg's servers, all in the same script. The visualizations Crazy Egg generates help publishers identify which parts of their website are being accessed frequently and which are being ignored. Publishers use this information to improve their web design and optimize their content.

[1] Not everyone knows latitude and longitude by heart. Luckily, Google has additional functions for converting street addresses to geographical coordinates. Learn more at http://code.google.com/apis/maps.

Figure 2 CrazyEgg's heatmap visualization highlights trafficked areas of publishers' websites

Crazy Egg's third-party script is considered a "passive" script; it records statistical data without any interaction from the publisher. The publisher is solely responsible for including the script on the page. The rest happens automatically.

Not all analytics scripts behave passively. MixPanel is an analytics company whose product tracks publisher-defined user actions to generate statistics about web site visitors or application users. Instead of generic web statistics, like page views or visitors, MixPanel has publishers define key application events they want to track. Some sample events are "user clicked the signup button" or "user played a video". Publishers write simple JavaScript code (see listing 2) to identify when the action takes place and then call a tracking method provided by MixPanel's third-party scripts to register the event with their service. MixPanel then assembles this data into interesting funnel statistics to help answer questions like, "What series of steps do users take before upgrading the product?"

Listing 2 Tracking user signups with the MixPanel JS API

```
<script src="http://api.mixpanel.com/site_media/js/api/mixpanel.js">
</script>

<script>
var mpmetrics = new MixpanelLib(PUBLISHER_API_TOKEN);          #1

$(function() {                                                 #2
    $('#signup-button').click(function() {
        mpmetrics.track("signup button clicked");
    });                                                        #3
});
</script>
```

#1 Initialize library
#2 Track click event with jQuery
#3 Submit event to MixPanel

Unlike Crazy Egg, MixPanel requires some development work by the publisher to define and trigger events. The upside is that the publisher can collect richer data and answer different questions about users' activities.

HTTP API wrappers

Analytics and metrics gathering scripts are examples of third-party scripts being used in ways that the client never sees or interacts with. There's another use case that works in a similar fashion: as a client-side wrapper for a third-party HTTP API.

Web APIs have become a must-have feature for growing web applications. They enable programmatic access to a web application by exposing HTTP endpoints that accept and/or return structured data, usually in XML or JSON formats. APIs enable outside developers to write code that hooks into your software, giving them the ability to add custom features, data export tools, or even mashups that blend a number of services together. Normally, HTTP APIs are accessed server-side using a scripting language like Ruby or PHP. But, as JavaScript has become more popular, increasingly we're seeing HTTP APIs being accessed from client code running on the browser as well.

There's a number of benefits to accessing web APIs from the web browser. As an API consumer, any code executing on the browser is code that doesn't have to run on your servers, which results in lower bandwidth and CPU costs. Secondly, there are many web sites that are completely static and don't rely on server-side code. If developers want their static web site to interact with a web application, their only recourse is through code that runs on the client. But, perhaps the biggest benefit is that JavaScript is the most widely used web programming language today—every website has it. If you're developing or maintaining an HTTP API and you want to grow a successful developer platform, you're going to want to support API access from the web browser.

Let's look at an example. You're probably familiar with LinkedIn, the popular social networking website for professionals. Apart from their website, LinkedIn has an HTTP API that enables developers to build applications on the LinkedIn platform. Additionally, LinkedIn distributes a third-party JavaScript library that enables developers to interact with their API directly from the browser. Developers can perform operations like query profile data for a user or look up business connections among users. It's a pretty powerful library; if you're not interested in straight-up data, it has methods for embedding fully-styled profile and company widgets on your website.

The following code example queries the LinkedIn API for the currently logged in user's list of connections and then uses jQuery (a popular JavaScript DOM library) to display that data on the page in an unordered list.

Listing 3 Displaying the logged in user's LinkedIn connections

```
<ul id="connections">       #1
</ul>

<script src="http://platform.linkedin.com/in.js">     #2
  api_key: YOUR_API_KEY
</script>

<script>
IN.API.Connections("me").result(function(connections) {     #3

    $.each(connections, function(profile) {
        var fullName = [profile.firstName, profile.lastName].join(' ');

        $('#connections').append('\
            <li>' + fullName + '</li>'
        );
    });
});
</script>
```

#1 Connections rendered here
#2 Requires LinkedIn API key
#3 Calls LinkedIn HTTP API

WHERE'S TYPE="TEXT/JAVASCRIPT"?

You might have noticed that these code examples don't include the attribute in script tag declarations. For an "type untyped" script tag, the default browser behavior is to treat it as JavaScript. It's your choice whether to include it or not!

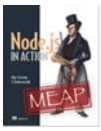
Nearly every large web application out there provides a JavaScript library for accessing their APIs. Facebook, Google, Twitter—you name it. It's a natural progression for a growing API to support client-side JavaScript. Yours should, too.

Summary

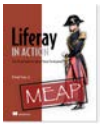
Third-party JavaScript is a powerful way of building embedded and highly distributable web applications. These applications can come in many shapes and sizes, but we looked at three specific use cases: as interactive widgets, as passive scripts that collect data, and as developer libraries that communicate with third-party web APIs.

Here are some other Manning titles you might be interested in:

Secrets of the JavaScript Ninja
John Resig and Bear Bibeault



Node.js in Action
Mike Cantelon and TJ Holowaychuk



Liferay in Action
The Official Guide to Liferay Portal Development
Richard Sezov, Jr

Last updated: September 12, 2011