The Event-Driven Data Layer

By Jim Gordon I 04/23/2019

Recently, I wrote an appeal to Adobe suggesting they implement first-party support for an event-driven data layer in Adobe Launch. I *specifically* mentioned an "event-driven, asynchronous data layer". Let's just call it the Event-Driven Data Layer (EDDL) to keep it simple. Since the article's release, I've had a number of good conversations on this topic. The consensus *seems* to be that this is the right direction. For the purposes of keeping the article a manageable length, I didn't go into too much detail of what an EDDL might look like. The goal of this article is to define what it is and why it's a good thing.

Before we begin, let's be clear about one thing. There's an EDDL and a CEDDL.

CEDDL = Customer Experience Digital Data Layer. Legacy W3C.

EDDL = Event-Driven Data Layer. What this article is about.

I tried to find a less confusing abbreviation. Unfortunately, this one made the most sense. They're both types of data layers. A good way to remember the difference is the CEDDL is 25% more cumbersome to spell and to implement. That's being generous. The Event-Driven Data Layer describes what you're implementing: a data layer that is constructed and transmitted to your TMS by events.

Let's first walk through a concept that seems obvious but very few people care to think about. That is this: **in a TMS, every tag is triggered by a some event**. That means your pageview is triggered on an event. That event might be TMS Library Loaded (Page Top), DOM Ready, Window Loaded, or any other indication that the page has loaded. These are all events that happen on a page or screen. They are as much of an event as a click, mouseover, form submission, or anything else.

A pageview is a *tag (*think Adobe Analytics beacon*)*. Viewing a page is an *event*. A custom link is a tag. Clicking a button is an event. Logging in is an event. Submitting a form is an event. That event might also be associated with some tags. Make sense? This is important because **we need to abstract the tool from the data**. Yes, you might choose to load Adobe Analytics pageview code when the window loads… but you COULD also choose to load a pageview on a click. The EDDL thinks independent of tool-specific definitions. Let's dive into why this is the preferred method.

Table of Contents

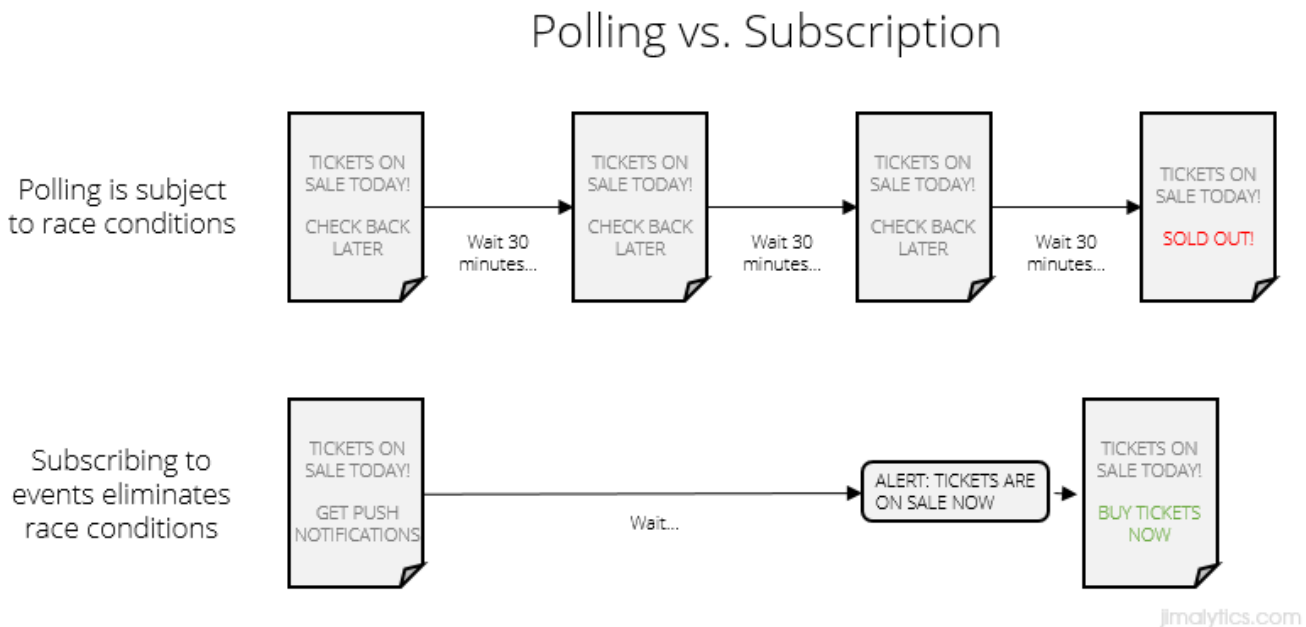
It’s harder to screw up

That doesn’t mean you *can’t* screw it up. It’s just harder to. You still need some code that sits in your header. The difference with the EDDL is:

1. It’s less code
2. You drop it in once and never have to touch it again

Maybe that means you’re simply declaring a variable (*var foo = [];*). Maybe it means you’re dropping in a little more code (copy/paste exercise). There’s no way around it – the variable or the function has to exist before you do anything with it. After that it’s all gravy. Timing is a non-issue with an EDDL. That’s because it proactively sends a message when a thing happens. Other data layer methodologies *poll* objects (like the Data Element Changed event). What does that mean?

Imagine you’re waiting on popular concert tickets to go on sale. You know what *day* they go on sale, but not what *time*. You know they’ll sell out fast, so you refresh Ticketmaster every half hour to see if their status changed. If you’re checking (or *polling*) to see if they’re for sale at 12:00 and they go on sale at 12:01, you might be out of luck. This is what’s called a *race condition*. After the status of the tickets change, you’re racing to see if you can get one before they’re gone.

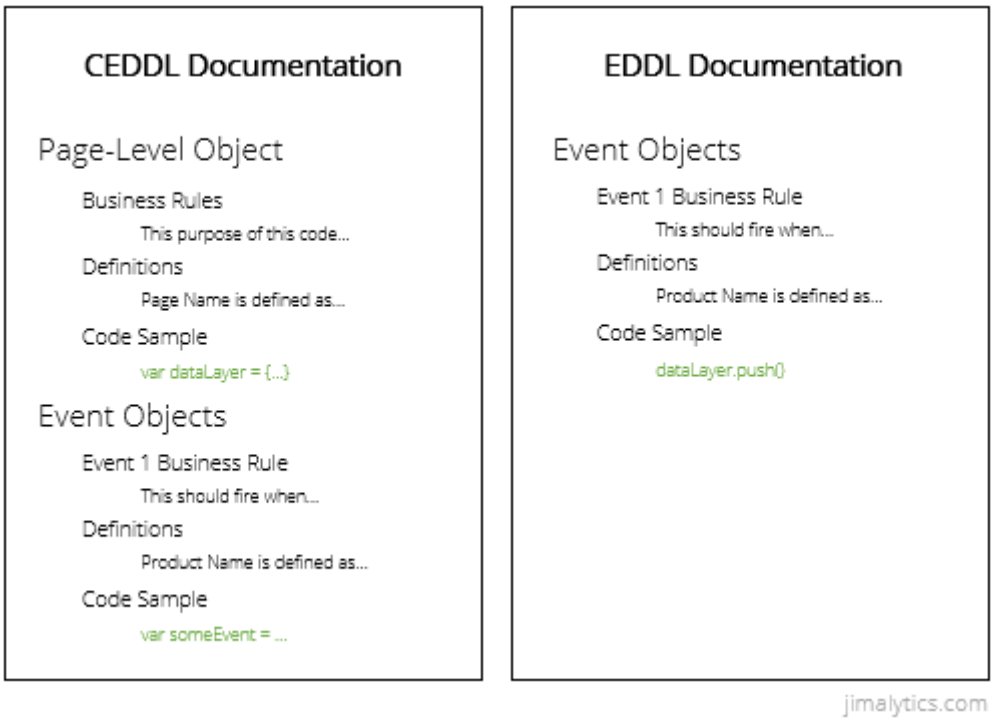


The same thing happens when you monitor data layers. If you go from Page A to Page B and you’re monitoring the object, there’s a very real chance you’ll be on the next page before your TMS realizes anything happened. That seriously sucks. You know what would fix this and save a lot of time?

Subscribing to push notifications. Proactively tell me when the tickets are on sale. Don't worry about loading the entire page object up-front. If we need to wait on servers, *let me know* when user information has propagated and then we'll trigger a pageview. The EDDL uses these push notifications so you don't have to worry about missing out on data.

It's easier to communicate

Prioritization of data layers is difficult when it *feels* like a lot of work and its value isn't immediately clear. Multiple code patterns paired with multiple sets of instructions *feel* like more work than one. We're already taking focus away from the value of the data layer at this point.



The CEDDL requires teams to learn multiple concepts/patterns. There's a page object… and then there are events. While it might be subconscious, multiple concepts/patterns (even simple ones) requires switching mental gears. They're also both explained as though they're different things. Here's an example from Adobe where both the W3C and another methodology are recommended. If I'm not very technical or I'm new to analytics, this would make my head spin.

The EDDL is much simpler. You explain it once, use the same code pattern, and can be easily dropped into a template. Here's how the documentation might look:

> **Data Layer Documentation**
> 1.0: Page and User Information
> Trigger: As soon as the information is available on each page loads or screen transition.
> Code: *dataLayer.push({"event":"Page Loaded","page":{…}…});*

> 1.1: Email Submit
> Trigger: When a user successfully submits an email address.
> Code: *dataLayer.push({“event”:“Email Submit”,“attributes”:{…}…});*

The documentation is consistent. We aren’t suggesting page stuff is different from event stuff. Remember that pageviews are triggered via events, too. You won’t send a pageview until you have the data you need, either. You’re not trying to figure out whether you can cram it into the header, before Page Bottom, ahead of DOM Ready, or before Window Loaded. We can just say: “*When the data is there, send the pageview.*” A lot of companies opted out of the W3C CEDDL for this reason alone.

It’s just as comprehensive

You can literally create the W3C schema with it. An effective EDDL has some kind of computed state I can access that functions like any other JSON object. What does a computed state mean, exactly? In the context of data layers, it means the content that was passed into it is processed into some kind of comprehensive object. Let’s pretend we’re using *dataLayer.push() and* I’m pushing information into the data layer about the page and the user. This is a Single Page App, so we will want to dynamically replace the name of the page as the user navigates. Similar to Google Tag Manager, this pushes the page and user data into the *dataLayer* object (because *dataLayer* makes more sense than *digitalData*):

```
> dataLayer.push({"page":{"pageName":"shoes page","pageCategory":"shoes"},"user":{"id":"ABCD123"}})
< 1
> dataLayer
< ▼[{…}]
    ▼0:
      ▼page:
          pageCategory: "shoes"
          pageName: "shoes page"
        ▶__proto__: Object
      ▼user:
          id: "ABCD123"
```

As a business user, it’s a bit of a stretch to learn how arrays work. I just want to see what the data looks like when the page loads so I know what I can work with. That’s where a computed state is useful. As a technical stakeholder, I can advise the business user to paste *dataLayer.computedState* into their console to see what data is available:

```
> dataLayer.computedState
{page: {…}, user: {…}}
  page:
    pageCategory: "shoes"
    pageName: "shoes page"
    __proto__: Object
  user:
    id: "ABCD123"
```

Looks like your average JSON object, right? Let's see what happens when we want to change a field.

Side note: In hindsight, I probably should have named the *pageCategory* and *pageName* fields simply "*category*" and "*name*". I'll save naming convention recommendations for another post…

Here we're just wanting to change the *pageName* and *pageCategory* fields. You can see the shoes data is still in that array above the hats page data. However, since that was the last information passed into the data layer, the computed state should update to reflect those changes:

```
> dataLayer.computedState
{page: {…}, user: {…}}
  page:
    pageCategory: "hats"
    pageName: "hats page"
    __proto__: Object
  user:
    id: "ABCD123"
```

There you have it. I should have the ability to add data and clear out fields, as well. For those who aren't as technical, **please note that this computed state stuff is NOT functionality baked into *dataLayer.push()* by default**. I did some extra work to manually create the *computedState* object. This is for modeling purposes only. Also note that a key differentiation between this and GTM's data

layer is a publicly exposed computed state. GTM does retain a computed state, but isn't (easily) accessible via your console.

With this functionality, the EDDL is more capable and accessible than any other client-side data layer technique.

Final Thoughts

One reason people don't implement data layers is because it's intimidating. You're part of a large organization with many stakeholders. Maintaining data layer standards takes work. You're right. Let's also acknowledge that maintaining *anything* is hard and requires a certain level of discipline. There's turnover on your team. Developers cycle in and out. Oh, by the way – you have to get this stuff prioritized, too!

The Event-Driven Data Layer is easier to document. It's easier to implement and not as vulnerable to timing issues. It's what a minority of sophisticated companies have already built for themselves (100 different ways) and what the majority needs to adopt. This data layer supports any schema you want. If you like how the W3C is structured, build it. If not, don't.

One critical piece you'll notice I didn't link you to some EDDL library. All of this information reflects how an EDDL *should* behave. There are many EDDLs out there and there won't likely be one single standard. However, Event-Driven Data Layers *will* eventually replace the CEDDL model. It makes more sense. I will commit to settling on a single recommendation in the coming months. There are a few examples out there:

1. Google Tag Manager
2. Data Layer Manager

If you have a public-facing event-driven data layer framework, add a comment or message me on Twitter and I would be happy to add it to this list. In the meantime, if you're working on a data layer – don't let the lack of a "standard" stop you from building one. If you want to use the CEDDL, go for it! *Having* a data layer is better than not having one.

Related Posts:

1. **Event-Driven Data Layer (EDDL) Demo**
2. **Top Event-Driven Data Layer Mistakes**
3. **Dream Team: EDDL and HTML5 Data Attributes**

Category: Governance Implementation Tag Management Tags: Adobe Launch, data layer, EDDL, event-driven data layer, javascript

14 thoughts on “The Event-Driven Data Layer”

Ally McDermid
07/05/2019

Thanks Jim, great article.
Is there a solution that doesn’t rely on any one tag manager / EDDL library?
For example if I wanted to switch to Launch from GTM, could launch pick up dataLayer.push({event: ‘form submit’});?

Jim Gordon Post author
07/11/2019

Hey Ally! Data Layer Manager is TMS agnostic. Outside of that, I’m not aware of any other EDDL.

Ally McDermid
07/15/2019

Cool, thanks again.

Pingback: Why You Need a Data Layer for Web Analytics Implementation I Blast Analytics

Casper Radil
10/15/2019

Interesting article. I build this type of EDDL datalayer handler for Ensighten (for GA), but I actually finds that it poses a lot of problems that I would rather live without (for example overwriting the

datalayer prototype object in order to prevent GTM overtaking the prototype if it is accidentially implemented on for example campaign sites etc.). What do you think about the difference between AA and GA in terms of implementing the EDDL?

Jim Gordon Post author
10/15/2019

Hey Casper, Thanks for reading! The EDDL should be tool-agnostic. After working with hundreds of implementations, I find it's equally as effective for both AA and GA.

Regarding GTM hijacking – I see that as a data governance issue. The real solution to that problem would be to prevent GTM from being deployed in the first place. A more temporary solution would be to rename your object to something other than dataLayer.

Pingback: #133: Server-Side vs. Client-Side Tracking with Mike Robins - The Digital Analytics Power Hour: Data and Analytics Podcast

Roger
11/02/2020

Hello Jim,
Thank you for the article, really well explained. I have a "stupid" question about how this data is sent to Adobe Analytics.
Implementing the EDDL explained way, pushing variables on the data layer on specific event (page view, click interaction, etc)… Once you have then the dataLayer with these new variable-values… How we should send this to Adobe Analytics? Add the tag manager script on the page and from the TMS add some kind of eventListner every time the datalayer gets populated send X to Adobe Analytics?
Thank you in advance!

Jim Gordon Post author
11/30/2020

Hey Roger – You would use an Adobe Launch extension like Data Layer Manager or Adobe Client Data Layer to send the data with Launch rules to Adobe Analytics.

Pingback: Third-party JavaScript concenrs I Snyk Blog

Nidhi Agrawal
03/05/2021

Hello Jim Gordon,
Thank you for the article. I have one question here how to unlock “Pro features” in Data Layer manager. As its mentioned in this link https://techdocs.searchdiscovery.com/adobe-solutions/adobe-launch/launch-extensions/data-layer-manager/extension-configuration/pro-mode-configurations ,that it can be unlock by Search Discovery Account Manager. But i am looking for required action/steps/process form my side to configure DLM and enable Pro features it its.

Pingback: AEP Web SDK 101 – Set up – Part 2 – The Learning Project

Pingback: Headless CMS with AEM: A Complete Guide I One Inside

Pingback: Using an event-driven data layer with Adobe Launch - Loop Horizon