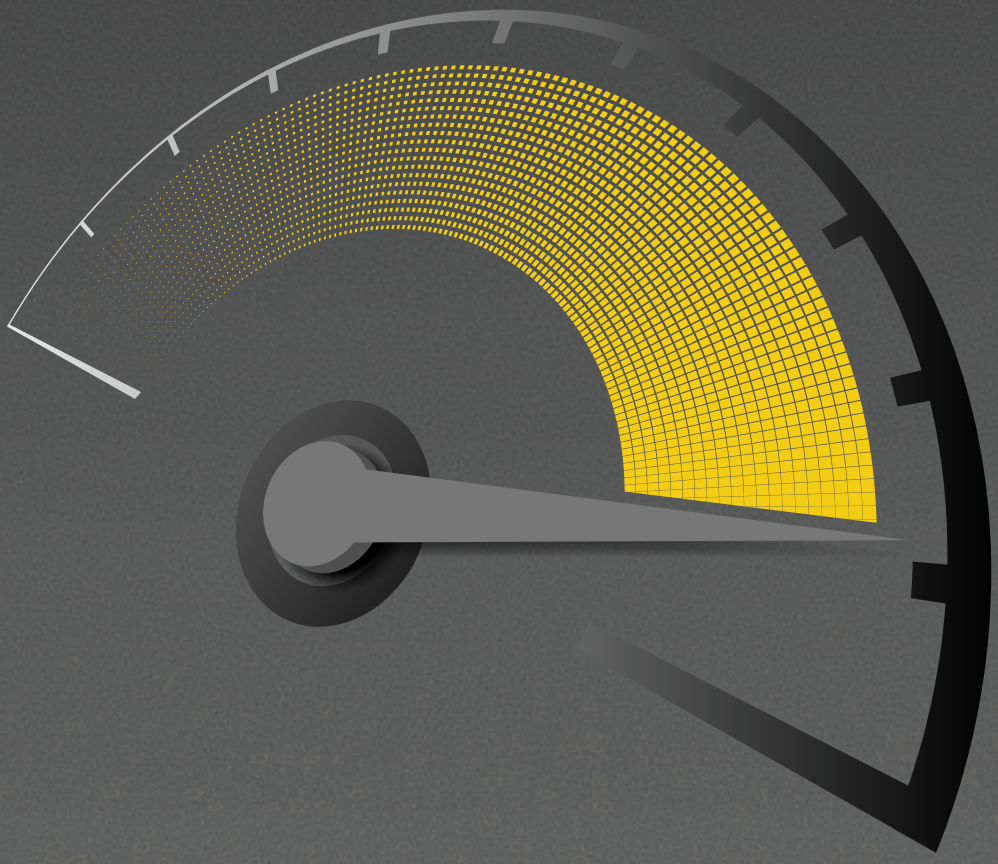
THE DATA LAYER

From Novice to Expert in 2.5 Seconds



RED BREAD | ObservePoint

Introduction

This white paper is a sports car.

You probably didn't know that when you downloaded it. Or that you could even download a sports car. Go figure.

This white paper is a sports car among white papers, and you're about to go 0 to 60 in your knowledge of the data layer. Moving from basic to advanced, you're going to learn about:

1. The fundamentals of the data layer
2. How to populate the data layer (including on a SPA site)
3. How to reference the data layer in your tag manager
4. How to use your data layer to simplify analytics testing

Buckle up.

Table of Contents

The Fundamentals

What is a data layer?

A data layer is a Javascript object with a consistent structure that you can reference throughout your digital property. Ideally it will have a common structure that is easily identifiable and standards driven.

Here's an analogy:

Think of a data layer as a single drop-off and pick-up location for parents sending their kids to an elementary school. Because the school wants to keep everything in order, they have parents drop children off at one location and then pick children up at the same location later that day. This keeps little Tommy from wandering off or school staff from wondering where and how kids are being picked up.

Comparably, the data layer is where you line up all your data so it can be picked up by the different technologies you deploy, both first-party and third-party.

Why do I need one?

With a data layer, you have a consistent place to put information that is publicly accessible and can be used for things like tags, pixels, analytics, or testing tools—like ObservePoint!

Ultimately, centralizing data collection into a single hub before distributing it out to different technologies has the following benefits:

- *Standardized data across channels*
- *Greater governance over data collection*
- *Easier extensibility of technology*

How do I implement a data layer?

Implementing a data layer is as simple as instantiating a new Javascript object to each page of your site. But how you implement the data layer (namely, the structure) is key.

There are competing standards surrounding data layers, and whichever one you use just has to be right for you. Having said that, working against a standard is a massive accelerator. The (most popular) W3C standard can be found here: Customer Experience Digital Data Layer 1.0

Google also has a data layer standard, called dataLayer, but there isn't necessarily a hard and fast standard (at least, not that I could find).

In this white paper, I used the W3C standard (the data layer is called digitalData) in conjunction with an Adobe Dynamic Tag Management implementation. You can find information about manipulating Google's data layer in their developer guide.

How and where do I instantiate?

Data layers should be instantiated (created) as early as possible in the page lifecycle. Data layers are used by tag management solutions such as Adobe Launch and Google Tag Manager, so your tag manager will fail if you reference a ubiquitous foundational data layer that isn't there.

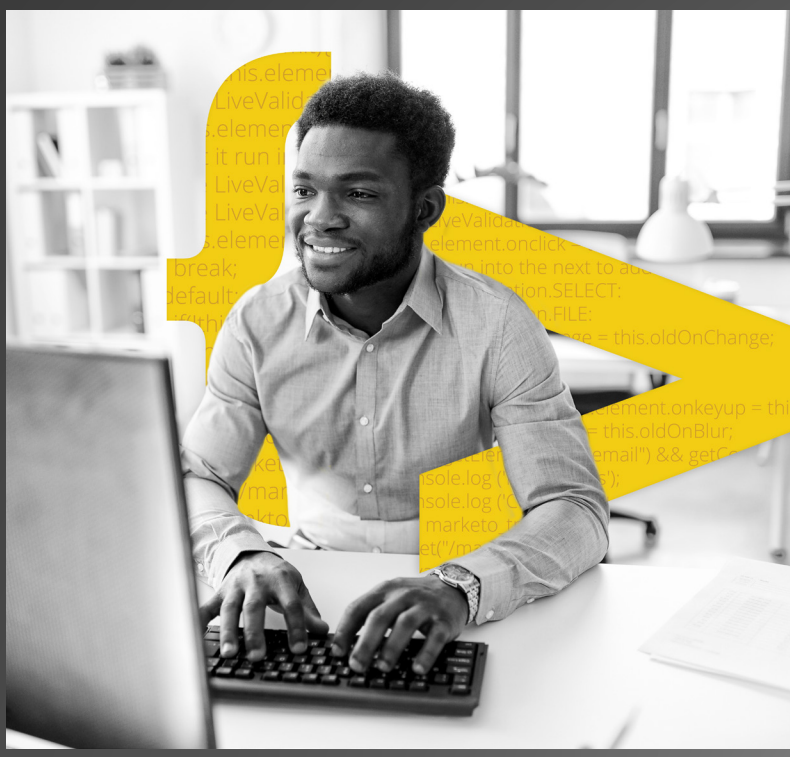
There are some differences between instantiating on a server-side application versus on a single-page application (SPA) site.

For a server-side application

When you're working with a server-side application, you're writing out the data layer directly to the page before the page is rendered. It captures the data from various sources, then dumps it onto the page in the form of a data layer object.

You're not going to change it often, since you're building it out almost completely with each page. A SPA site allows for more flexibility that way.

There will be more on this topic of server-side data layers in the section below about populating the data layer.



For a SPA site

You instantiate the data layer for your SPA site with a simple statement at the tippity-top of the head, like so:

```
window['digitalData'] = { events: [] };
```

You want to load the data layer as early as possible because everything is happening asynchronously, so the earlier the better.

When you call this digital data layer, it's empty, but it has a basic structure that you've defined in a global class elsewhere. Whenever someone does a routing change, it creates the objects of that class anew, based on whatever new information you have, overwriting what was in there before. So you can start small with very little information (literally nothing), and as you gather more data, add to the data layer.

Obviously, you put in a lot more data later, but this means that you do not have to write anything about the presence of the layer itself.

What does a data layer look like?

Here is a sample of what a data layer might look like. I created it from the perspective of a product page to show what that part populates:


```
{
  "page": {
    "pageInfo": {
      "geoRegion": "US",
      "sysEnv": "MyServer"
    },
    "category": {
      "pageTemplate": "/product",
      "pageType": "Product",
      "merchandisingCategory": {
        "catID": "charlie-category",
        "subCatID": "charlie-subcategory"
      }
    },
    "attributes": {
      "errorCode": null,
      "deviceType": "desktop"
    }
  },
  "user": {
    "profile": {
      "profileInfo": {
    "profileID":"47dcebcc-9860-467e-95da-d71d1273898e",
        "returningStatus": "returning",
        "type": "non-member"
      }
    },
    "attributes": {
      "isSignedIn": false
    }
  },
  "product": [
    {
      "productInfo": {
        "productID": "xyz123",
        "productName": "Charlie Product Name"
      },
      "attributes": {
        "ratingValue": 4.8217,
        "reviewCount": 785
      }
    }
  ],
  "cart": {
    "cartID": "50d90773-054c-44b9-9baa-4e01141aa211",
    "promoCode": "SAVEMORE",
    "numItems": 2,
    "price": {
      "cartTotal": 147.23,
      "specialOfferCartTotal": 105.63,
      "basePrice": 0,
      "currency": "USD",
      "shipping": 0,
      "transactionTotal": 110.91,
      "tax": 5.28,
      "allowances": 41.6,
      "item": [
```

```
{
    "product": {
      "productInfo": {
        "productID": "abc987",
        "productName": "Alpha Product"
      }
    },
    "quantity": 1,
    "price": {
      "cartTotal": 47.24,
      "specialOfferCartTotal": 35.64,
      "basePrice": 47.24,
      "currency": "USD",
      "shipping": 0,
      "tax": 1.78,
      "allowances": 11.6
    },
    "category": {
      "catID": "alpha-category",
      "subCatID": "alpha-subcategory"
    }
  },
  {
    "product": {
      "productInfo": {
        "productID": "def456",
        "productName": "Beta Product"
      }
    },
    "quantity": 1,
    "price": {
      "cartTotal": 99.99,
      "specialOfferCartTotal": 69.99,
      "basePrice": 99.99,
      "currency": "USD",
      "shipping": 0,
      "tax": 0,
      "allowances": 30
    },
    "category": {
      "catID": "beta-category",
      "subCatID": "beta-subcategory"
    }
  }
 ]
 }
 },
 "transaction": {
  "transactionID": null,
  "promoCode": null,
  "total": {
    "cartTotal": 0,
    "specialOfferCartTotal": 0,
    "basePrice": 0,
    "currency": null,
    "shipping": 0,
    "transactionTotal": 0,
    "tax": 0,
    "allowances": 0
  },
  "profile": {
    "address": {
      "stateProvince": null,
      "country": null
    }
  }
 },
 "event": [],
 "version": "1.0",
 "lastEvent": {}
}
```


We can see that in this fictional data layer, the user has added two items to his cart previously and has applied a promotion code which has given allowances. As the user moves through the checkout process and gets to the order complete stage, the information in the cart object will be moved into the transaction object, and a transaction ID given.

Notice that basic instantiations of objects are done, even when not in use—the above transaction is a good example of this. By making the empty object, you do not have to check for its presence beyond the broadest try catch on the tag.

With that done, we can easily implement analytics and tagging.

Populating the Data Layer

Methods for pushing data into the data layer

The data layer will be filled from server-side code in the case of a traditional web page, and potentially modified as data on the page changes.

My recommended methodology is to make a model structure that matches what you want to see, dummy it out in json with false data, and then use an online converter such as http://json2csharp.com/ to make the classes.

I would then drop those into the project, populate them from my code and when I am ready to drop them into a page use something like JsonConvert.SerializeObject. This example is broadly C#-based but very similar methodologies can be employed for other server-side languages.

For a SPA site, it will all be modified as the page changes, which poses difficulties to keep it up to date. But it is essential that you do so because you will use the data layer as a form of state for third parties.

There is no cookie cutter way to populate the data layer—how you do so will ultimately depend on the nature of your website. Below are some of the most common details that people capture in their data layers:

* *Page data (template, obfuscated server name, browser type)*
* *User data (non-PII: first name, country, state/province)*
* *Product data (id, name, price, sale price, category)*
* *Cart data (ids, prices, sale prices, categories)*
* *Event data (routings/url changes, goal clicks)*
* *Transaction data (order number, ids, prices, sale prices, categories)*

Special Note About Security

Never put into a data layer anything private. It is freely scrapable by competing companies. This includes PII, so emails, addresses, etc.—all should AT MINIMUM be hashed. Personally, I refuse to give that info to third parties, and a lot of tag companies will ask for it.

Likewise, never put info into a page that you wouldn't want a competitor to know. The most common infractions are data about product profit and inventory. If you want a second party to have this info (and it is useful for things like weighting product recommendations), then make it available by a data feed.

Below are a couple negative scenarios that could result from including sensitive information in your data layer:

1. A competitor could buy a ton of a product that is operating at a loss to drive down the profits of your company.

2. A competitor could automatically purchase and cancel orders for low inventory products to cause stock outs.

Referencing the Data Layer in Your Tag Manager

Data layers make passing data to your tags much, much easier. We'll dig into Adobe Dynamic Tag Management (DTM) specifically, but the same principles apply across tag managers.

Adobe Dynamic Tag Management (DTM)

In DTM there are several ways you can grab data from the page and push it into a data element. Data elements are what you map to variables for each of your vendors. In DTM, you can technically grab data from any of the following:

- A URL parameter
- An HTML element using a CSS selector
- A cookie
- **A Javascript object**
- **A custom script**

But when it comes to the data layer, you only need to worry about the last two.

Referencing the data layer for data elements

Once you've mapped data layer properties to data elements in DTM, you can set your eVars, props, and other analytics variables equal to those data elements.

As a JS Object

Having built out your data layer, creating most data elements in DTM is easy-peasy. Just create a new data element, name it, and use standard Javascript syntax to reference the data within the data layer object, such as 'digitalData.cart.cartID'.

As a custom script

In some cases, you may need to further manipulate the data in the data layer before including it as a data element. In this case, you would use a custom script, which would allow you to manipulate data to be in a certain format.

One example is when you're working with a Criteo or Coherent pixel and you need to grab all the cart items from the data layer and build out an array in a format the pixel can read. The code below is an example of formatting data pulled from the data layer for a Coherent pixel:

```
window._cp=function(){(window._cp.queue=window._cp.queue|| []).push(arguments)},function(){var a=document.createElement("script");a.type="text/javascript",a.async=1,a.src=("https:"==document.location.protocol?"https://":"http://")+"api.coherentpath.com/tracker/v1/orvis/analytics.js";var b=document.getElementsByTagName("script")[0];b.parentNode.insertBefore(a,b)}();
var items = _satellite.getVar('window.digitalData.transaction.item') || [];
var itemIds = [];
var itemQtys = [];
var itemAmts = [];
var transactionIds = [];
if(_satellite.isArray(items)&&items.length>0){
  items.forEach(function(item) {
    itemIds.push(item.product.productInfo.productID);
    itemQtys.push(item.quantity);
        itemAmts.push(item.price.basePrice-item.price.allowances);
    transactionIds.push(_satellite.getVar('order id'));
  });
}
window._cp('recordTrans', {
sku: itemIds.join('|'),
quantity: itemQtys.join('|'),
amount: itemAmts.join('|'),
transaction_id: transactionIds.join('|')
})
```

Note: In Adobe Launch you wouldn't use the satellite.getVar() method. You would just reference the data layer object directly like this: 'window.digitalData.cart.item'.

Pro Tip:

The Custom Script option gives you a lot of flexibility for creating data elements (which will later be passed to your tags). Something you might wonder: couldn't I just use a custom script to scrape data directly from my site instead of pushing into the data layer first and then into DTM via the data layer?

A situation where you might be tempted to do so would be if you had a single vendor that needed a data point from your site. You think, "Better to avoid the data layer altogether and just push the data straight to the vendor."

Avoid this temptation like the plague! Think about this: what happens if the structure of your website changes and you're no longer able to use the same Javascript/jQuery selectors to scrape data from your site? Your tags won't have access to data anymore.

Pushing data into a data layer and then referencing that data layer is much less likely to break. Plus, if later you have a new vendor that needs access to that data, it will already be built into the data layer for you to use. Much better!

Using Data Elements

Once you've created a data element, you can use it for multiple purposes.

First, you can use it to define page load rules for tag firing. Because data elements will load before tags, you can use the value(s) of a data element to determine whether or not a tag should fire. So you can use your data layer to determine when tags should fire. So cool!

Second, the data elements serve as the containers for your variables to reference for data. This allows you to standardize the data being passed to all your martech vendors.

Assigning data elements to Adobe Analytics and Adobe Target

Equipped with your data elements, setting up your global variables for Adobe Analytics and Adobe Target is extremely simple. For Adobe Analytics, just select the eVar or prop name, and then drop in the data element name surrounded by the "%" symbol, which is Adobe's shortcode way of indicating a data element.

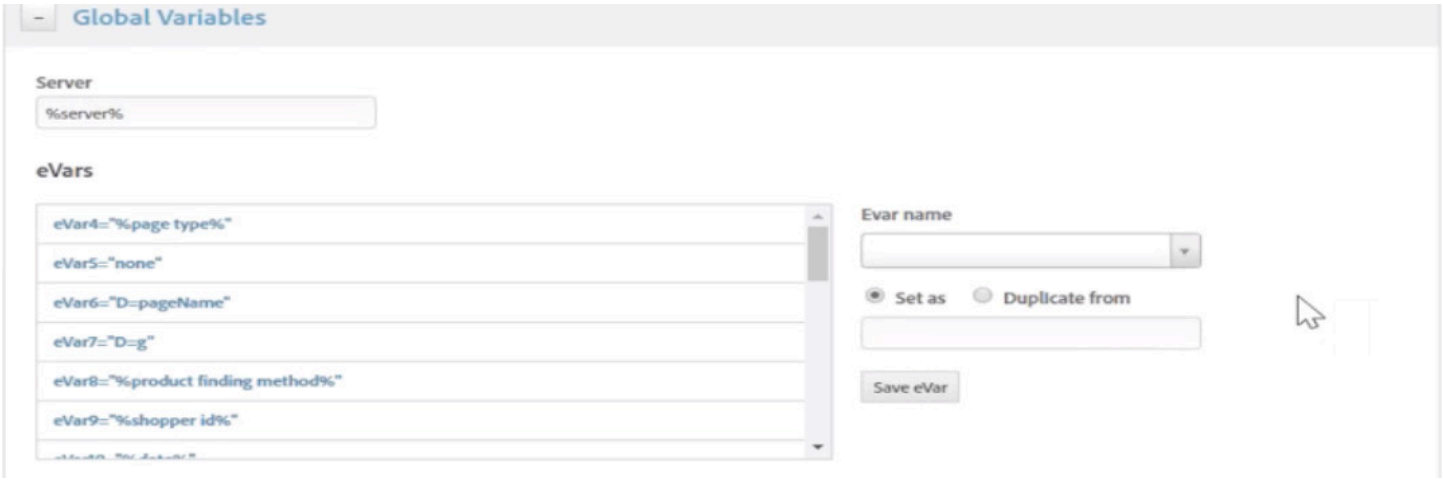
Assigning data elements to third-party tags

You can also use data elements for third-party tags using the Code method. That's what we've done with our Criteo tag at Orvis.

```
window.criteo_q = window.criteo_q || [];
window.criteo_q.push (
        {
        event: "track transaction",
        id: _satellite.getVar('order id'),
        item: _satellite.getVar('order items')
        },
        {
)
```

Just set the appropriate variable equal to your data element like we showed above.

And that's it! You've successfully integrated your data layer with your tag manager. Go ahead and take the rest of the day off, because the investment you've made in your data layer is going to pay some high dividends. (But make sure to come back, because we're not done yet.)



Simplifying Analytics Testing with the Data Layer

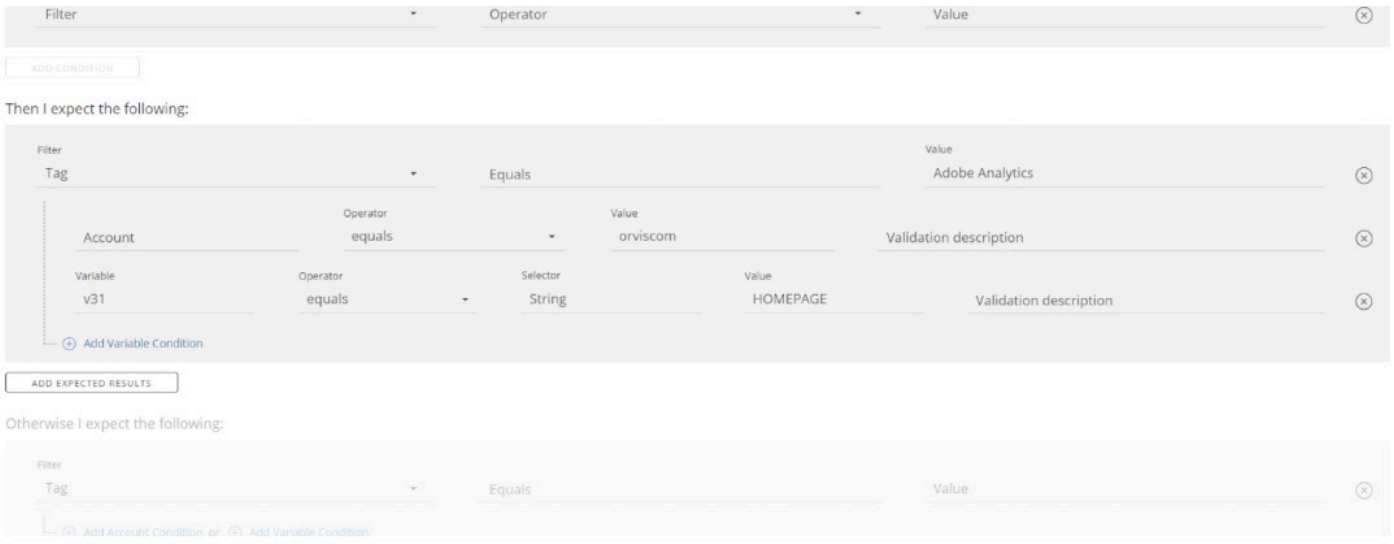
One of the biggest struggles of collecting data for analytics and martech is keeping things consistent. As your website grows, different teams may stray from standard conventions for naming and populating variables. That creates problems for reporting and decision-making.

Thankfully, if you've correctly implemented and adopted the data layer, you not only have a central data hub for your tracking technologies, but you can also centralize your analytics testing!

A good analogy is to think of automobile recalls: if car companies can identify issues in a prototype, then they won't have to worry about a massive recall process should that issue get passed on to their manufactured models. Such it is with finding errors directly in the data layer, as you have a single location to test for errors.

In the work I do for Orvis, we use ObservePoint's Rules feature to validate our data layer values. Below is an example where the pagename is being passed to Adobe Analytics (in eVar31) and then ObservePoint is looking for it in a custom rule—in this case to verify that the base url has a page name of **HOMEPAGE**.

After setting this rule, I don't have to worry because I'll receive a notification if anything ever goes wrong.



THE FINISH LINE

While it probably took you more than two and a half seconds to read this white paper, I ultimately kept my promise. You've moved from a basic understanding of the data layer to a clearer idea of what it takes to implement one.

Putting these ideas into practice will take some effort. But implementing a robust data layer will change the way you collect data. It will be a pivotal point in your company's analytics maturity, giving you that nitro boost to cross the finish line ahead of the competition.

If you need help getting started, visit Red Bread Lab for assistance and ObservePoint to help you catalog your data collection for an easier path to implementation.

SCHEDULE A DEMO

CONTACT RED BREAD LAB

About the Author

MARTY VANZWIETERING

Marten vanZwietering has been working in the technical space for the last two decades, programming for a variety of retail, hospitality and marketing strategy firms. Marty currently works full-time as a web development manager and architect at The Orvis Company, an outdoor recreation and sporting goods brand, and also consults part time for a design and data company named Red Bread Lab. Marty also loves to write – his first book, *My Unlife, a Zombie Autobiography* is available now, and should be purchased immediately.