

## **Module 6 - Testing, validation, and troubleshooting**

This toolkit is designed for Professional Developer Exam Aspirants. There are **six** Modules. Study Each module per week to stick to schedule. Technical Parts of applications are depicted in Videos, you can learn more about them from experience League. You can visit [Get prep page](#) to understand the contents and anticipate the learning journey.

This is Professional Exam, Developer toolkit Module 6. This module contains five sections.

### **6.1 JavaScript error reference**

Below, you'll find a list of errors which are thrown by JavaScript. These errors can be a helpful debugging aid, but the reported problem isn't always immediately clear. The pages below will provide additional details about these errors. Each error is an object based upon the [Error](#) object, and has a name and a message.

Errors displayed in the Web console may include a link to the corresponding page below to help you quickly comprehend the problem in your code.

For a beginner's introductory tutorial on fixing JavaScript errors, see [What went wrong? Troubleshooting JavaScript](#).

#### [List of errors](#)

In this list, each page is listed by name (the type of error) and message (a more detailed human-readable error message). Together, these two properties provide a starting point toward understanding and resolving the error. For more information, follow the links below!

- [Error: Permission denied to access property "x"](#)
- [InternalError: too much recursion](#)
- [RangeError: BigInt division by zero](#)
- [RangeError: BigInt negative exponent](#)
- [RangeError: argument is not a valid code point](#)
- [RangeError: invalid array length](#)
- [RangeError: invalid date](#)
- [RangeError: precision is out of range](#)
- [RangeError: radix must be an integer](#)
- [RangeError: repeat count must be less than infinity](#)
- [RangeError: repeat count must be non-negative](#)
- [RangeError: x can't be converted to BigInt because it isn't an integer](#)
- [ReferenceError: "x" is not defined](#)
- [ReferenceError: assignment to undeclared variable "x"](#)
- [ReferenceError: can't access lexical declaration 'X' before initialization](#)
- [ReferenceError: deprecated caller or arguments usage](#)
- [ReferenceError: reference to undefined property "x"](#)

- [SyntaxError: "0"-prefixed octal literals and octal escape seq. are deprecated](#)
- [SyntaxError: "use strict" not allowed in function with non-simple parameters](#)
- [SyntaxError: "x" is a reserved identifier](#)
- [SyntaxError: JSON.parse: bad parsing](#)
- [SyntaxError: Unexpected '#' used outside of class body](#)
- [SyntaxError: Unexpected token](#)
- [SyntaxError: Using //@ to indicate sourceURL pragmas is deprecated. Use //# instead](#)
- [SyntaxError: a declaration in the head of a for-of loop can't have an initializer](#)
- [SyntaxError: applying the 'delete' operator to an unqualified name is deprecated](#)
- [SyntaxError: await is only valid in async functions, async generators and modules](#)
- [SyntaxError: cannot use `??` unparenthesized within `||` and `&&` expressions](#)
- [SyntaxError: continue must be inside loop](#)
- [SyntaxError: for-in loop head declarations may not have initializers](#)
- [SyntaxError: function statement requires a name](#)
- [SyntaxError: getter and setter for private name #x should either be both static or non-static](#)
- [SyntaxError: identifier starts immediately after numeric literal](#)
- [SyntaxError: illegal character](#)
- [SyntaxError: invalid BigInt syntax](#)
- [SyntaxError: invalid assignment left-hand side](#)
- [SyntaxError: invalid regular expression flag "x"](#)
- [SyntaxError: label not found](#)
- [SyntaxError: missing \) after argument list](#)
- [SyntaxError: missing \) after condition](#)
- [SyntaxError: missing : after property id](#)
- [SyntaxError: missing ; before statement](#)
- [SyntaxError: missing = in const declaration](#)
- [SyntaxError: missing \] after element list](#)
- [SyntaxError: missing formal parameter](#)
- [SyntaxError: missing name after . operator](#)
- [SyntaxError: missing variable name](#)
- [SyntaxError: missing } after function body](#)
- [SyntaxError: missing } after property list](#)
- [SyntaxError: redeclaration of formal parameter "x"](#)
- [SyntaxError: return not in function](#)
- [SyntaxError: test for equality \(==\) mistyped as assignment \(=\)?](#)
- [SyntaxError: unlabeled break must be inside loop or switch](#)
- [SyntaxError: unparenthesized unary expression can't appear on the left-hand side of '\\*\\*'](#)
- [SyntaxError: unterminated string literal](#)
- [TypeError: "x" has no properties](#)
- [TypeError: "x" is \(not\) "y"](#)
- [TypeError: "x" is not a constructor](#)
- [TypeError: "x" is not a function](#)
- [TypeError: "x" is not a non-null object](#)
- [TypeError: "x" is read-only](#)
- [TypeError: 'x' is not iterable](#)
- [TypeError: More arguments needed](#)
- [TypeError: Reduce of empty array with no initial value](#)

- [TypeError: X.prototype.y called on incompatible type](#)
- [TypeError: can't assign to property "x" on "y": not an object](#)
- [TypeError: can't convert BigInt to number](#)
- [TypeError: can't convert x to BigInt](#)
- [TypeError: can't define property "x": "obj" is not extensible](#)
- [TypeError: can't delete non-configurable array element](#)
- [TypeError: can't redefine non-configurable property "x"](#)
- [TypeError: cannot use 'in' operator to search for 'x' in 'y'](#)
- [TypeError: cyclic object value](#)
- [TypeError: invalid 'instanceof' operand 'x'](#)
- [TypeError: invalid Array.prototype.sort argument](#)
- [TypeError: invalid assignment to const "x"](#)
- [TypeError: property "x" is non-configurable and can't be deleted](#)
- [TypeError: setting getter-only property "x"](#)
- [URIError: malformed URI sequence](#)
- [Warning: -file- is being assigned a //# sourceMappingURL, but already has one](#)
- [Warning: unreachable code after return statement](#)

1.

1. [InternalError: too much recursion](#)
2. [RangeError: argument is not a valid code point](#)
3. [RangeError: BigInt division by zero](#)
4. [RangeError: BigInt negative exponent](#)
5. [RangeError: invalid array length](#)
6. [RangeError: invalid date](#)
7. [RangeError: precision is out of range](#)
8. [RangeError: radix must be an integer](#)
9. [RangeError: repeat count must be less than infinity](#)
10. [RangeError: repeat count must be non-negative](#)
11. [RangeError: x can't be converted to BigInt because it isn't an integer](#)
12. [ReferenceError: "x" is not defined](#)
13. [ReferenceError: assignment to undeclared variable "x"](#)
14. [ReferenceError: can't access lexical declaration 'X' before initialization](#)
15. [ReferenceError: deprecated caller or arguments usage](#)
16. [ReferenceError: reference to undefined property "x"](#)
17. [SyntaxError: "0"-prefixed octal literals and octal escape seq. are deprecated](#)
18. [SyntaxError: "use strict" not allowed in function with non-simple parameters](#)
19. [SyntaxError: "x" is a reserved identifier](#)
20. [SyntaxError: a declaration in the head of a for-of loop can't have an initializer](#)

21. [SyntaxError: applying the 'delete' operator to an unqualified name is deprecated](#)
22. [SyntaxError: await is only valid in async functions, async generators and modules](#)
23. [SyntaxError: cannot use `??` unparenthesized within `||` and `&&` expressions](#)
24. [SyntaxError: continue must be inside loop](#)
25. [SyntaxError: for-in loop head declarations may not have initializers](#)
26. [SyntaxError: function statement requires a name](#)
27. [SyntaxError: getter and setter for private name #x should either be both static or non-static](#)
28. [SyntaxError: identifier starts immediately after numeric literal](#)
29. [SyntaxError: illegal character](#)
30. [SyntaxError: invalid assignment left-hand side](#)
31. [SyntaxError: invalid BigInt syntax](#)
32. [SyntaxError: invalid regular expression flag "x"](#)
33. [SyntaxError: JSON.parse: bad parsing](#)
34. [SyntaxError: label not found](#)
35. [SyntaxError: missing ; before statement](#)
36. [SyntaxError: missing : after property id](#)
37. [SyntaxError: missing \) after argument list](#)
38. [SyntaxError: missing \) after condition](#)
39. [SyntaxError: missing \] after element list](#)
40. [SyntaxError: missing } after function body](#)
41. [SyntaxError: missing } after property list](#)
42. [SyntaxError: missing = in const declaration](#)
43. [SyntaxError: missing formal parameter](#)
44. [SyntaxError: missing name after . operator](#)
45. [SyntaxError: missing variable name](#)
46. [SyntaxError: redeclaration of formal parameter "x"](#)
47. [SyntaxError: return not in function](#)
48. [SyntaxError: test for equality \(==\) mistyped as assignment \(=\)?](#)
49. [SyntaxError: Unexpected '#' used outside of class body](#)
50. [SyntaxError: Unexpected token](#)
51. [SyntaxError: unlabeled break must be inside loop or switch](#)
52. [SyntaxError: unparenthesized unary expression can't appear on the left-hand side of `!\*\*!`](#)
53. [SyntaxError: unterminated string literal](#)
54. [SyntaxError: Using `//@` to indicate sourceURL pragmas is deprecated. Use `//#` instead](#)

55. [TypeError: 'x' is not iterable](#)
56. [TypeError: "x" has no properties](#)
57. [TypeError: "x" is \(not\) "y"](#)
58. [TypeError: "x" is not a constructor](#)
59. [TypeError: "x" is not a function](#)
60. [TypeError: "x" is not a non-null object](#)
61. [TypeError: "x" is read-only](#)
62. [TypeError: can't assign to property "x" on "y": not an object](#)
63. [TypeError: can't convert BigInt to number](#)
64. [TypeError: can't convert x to BigInt](#)
65. [TypeError: can't define property "x": "obj" is not extensible](#)
66. [TypeError: can't delete non-configurable array element](#)
67. [TypeError: can't redefine non-configurable property "x"](#)
68. [TypeError: cannot use 'in' operator to search for 'x' in 'y'](#)
69. [TypeError: cyclic object value](#)
70. [TypeError: invalid 'instanceof' operand 'x'](#)
71. [TypeError: invalid Array.prototype.sort argument](#)
72. [TypeError: invalid assignment to const "x"](#)
73. [TypeError: More arguments needed](#)
74. [TypeError: property "x" is non-configurable and can't be deleted](#)
75. [TypeError: Reduce of empty array with no initial value](#)
76. [TypeError: setting getter-only property "x"](#)
77. [TypeError: X.prototype.y called on incompatible type](#)
78. [URIError: malformed URI sequence](#)
79. [Warning: -file- is being assigned a //# sourceMappingURL, but already has one](#)
80. [Warning: unreachable code after return statement](#)

## 6.2 [Analytics Tech Notes](#)

This guide provides helpful information on topics that don't belong to a specific analytics tool or component.

Adobe Analytics is a web analytics solution that enables businesses to measure, analyze and optimize their digital marketing activities. It is a cloud-based platform that provides real-time insights into website and mobile app performance, customer behavior, and marketing campaign effectiveness.

The platform uses a variety of data sources, including web and mobile app data, third-party data, offline data, and CRM data, to deliver a complete view of customer interactions across multiple

channels. Adobe Analytics offers a range of features that enable businesses to make data-driven decisions and drive digital transformation.

Some of the main features of Adobe Analytics include:

- Real-time data: Adobe Analytics provides real-time insights into customer behavior, enabling businesses to react quickly to changes in customer preferences or market trends.
- Segmentation: The platform allows businesses to segment their customer base by demographics, behavior, location, and other factors, making it easier to identify opportunities and target specific customer groups with personalized marketing campaigns.
- Data visualization: Adobe Analytics provides a range of data visualization tools, most prominently charts and graphs in Analysis Workspace and Analytics dashboards, which make it easy to interpret complex data sets and identify trends.
- Advanced analytics: The platform offers advanced analytics capabilities, including predictive analytics, machine learning, and AI-powered insights, which enable businesses to uncover hidden patterns and gain a deeper understanding of customer behavior.
- Attribution: Adobe Analytics includes attribution modeling tools that help businesses understand the impact of their marketing campaigns across different channels and touchpoints, enabling them to optimize their marketing spend and improve ROI.
- Reporting: The platform offers a range of reporting options, including scheduled reports, ad hoc reports, and customizable dashboards, which enable businesses to share insights with stakeholders and collaborate on data-driven decision making.

In summary, Adobe Analytics is a powerful web analytics solution that provides businesses with the tools they need to measure, analyze, and optimize their digital marketing activities. With real-time data, advanced analytics, and attribution modeling, businesses can make data-driven decisions that drive digital transformation and improve ROI.

[Here](#) is a video overview of Adobe Analytics.

## 6.3 [linkDownloadFileTypes](#)

When `trackDownloadLinks` (AppMeasurement) or `clickCollectionEnabled` (Web SDK) is enabled and a visitor clicks on a link, AppMeasurement checks the URL of the link for filetype extensions. If the link URL contains a matching filetype, a download link image request is automatically sent.

Use `linkDownloadFileTypes` to customize what file extensions you want to count as download links.

## NOTE

Only actual clicks are tracked automatically. The following types of links are not automatically tracked:

- File downloads that start automatically when a page loads
- Downloads that trigger after a redirect
- Right-clicking and selecting 'Save Target As...'
- Links that use JavaScript, such as `javascript:openLink()`

For these download types, you can manually send a [link tracking](#) call.

If a clicked link matches both exit link and download link criteria, the download link type takes priority.

## Download link qualifier using the Web SDK extension

The Download link qualifier text field uses regex to determine if a clicked link qualifies to be a download link.

1. Log in to [Adobe Experience Platform Data Collection](#) using your AdobeID credentials.
2. Click the desired tag property.
3. Go to the Extensions tab, then click the **Configure** button under Adobe Experience Platform Web SDK.
4. Under Data Collection, set the desired value in the **Download link qualifier** text field.

## Download link qualifier manually implementing the Web SDK

[Configure](#) the SDK using `downloadLinkQualifier`. The field uses regex on the clicked URL to determine if it is a valid download link. If `downloadLinkQualifier` is not defined, the default value is set

to `\\. (exe|zip|wav|mp3|mov|mpg|avi|wmv|pdf|doc|docx|xls|xlsx|ppt|pptx)$`.

```
alloy("configure", {
  "downloadLinkQualifier":
  "\\.(exe|zip|wav|mp3|mov|mpg|avi|wmv|pdf|doc|docx|xls|xlsx|ppt|pptx)$"
});
```

Copy

Toggle Text Wrapping

## Download Extensions using the Adobe Analytics extension

Download Extensions is a list of file extensions with a field to add more under the Link Tracking accordion when configuring the Adobe Analytics extension.

1. Log in to [Adobe Experience Platform Data Collection](#) using your AdobeID credentials.
2. Click the desired tag property.
3. Go to the Extensions tab, then click the **Configure** button under Adobe Analytics.
4. Expand the Link Tracking accordion, which reveals the **Download Extensions** field.

Add file extensions to the list by entering text in the field and clicking **Add**. Remove file extensions from the list by clicking their respective 'X' icon.

### s.linkDownloadFileTypes in AppMeasurement and the Analytics extension custom code editor

The `s.linkDownloadFileTypes` variable is a string of comma-separated file extensions. Do not use spaces.

If this variable is not defined, automatic download link tracking does not work (even if `trackDownloadLinks` is true).

```
s.linkDownloadFileTypes =  
"doc,docx,eps,jpg,png,svg,xls,ppt,pptx,pdf,xlsx,tab,csv,zip,txt,vsd,vxd,xml,js,  
,css,rar,exe,wma,mov,avi,wmv,mp3,wav,m4v";
```

Copy

Toggle Text Wrapping

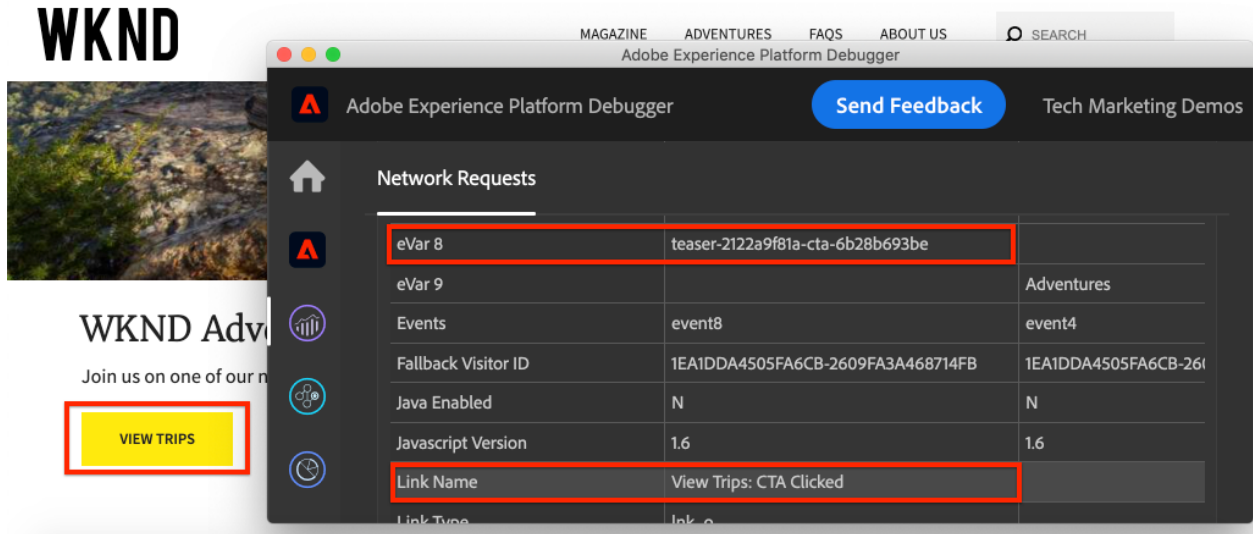
## 6.4 Track clicked component with Adobe Analytics

Use the event-driven [Adobe Client Data Layer with AEM Core Components](#) to track clicks of specific components on an Adobe Experience Manager site. Learn how to use rules in the tag property to listen for click events, filter by component and send the data to an Adobe Analytics with a track link beacon.

What you are going to build

The WKND marketing team is interested in knowing which `Call to Action` (CTA) buttons are performing the best on the home page. In this tutorial, let's add a rule to the tag property that listens for the `cmp:click` events from **Teaser** and **Button** components. Then send the component ID and a new event to Adobe Analytics alongside the track link beacon.





## Objectives

1. Create an event-driven rule in the tag property that captures the `cmp:click` event.
2. Filter the different events by component resource type.
3. Set the component id and send an event to Adobe Analytics with the track link beacon.

## Prerequisites

This tutorial is a continuation of [Collect page data with Adobe Analytics](#) and assumes that you have:

- A **Tag property** with the [Adobe Analytics extension](#) enabled
- **Adobe Analytics** test/dev report suite ID and tracking server. See the following documentation for [creating a report suite](#).
- [Experience Platform Debugger](#) browser extension configured with your tag property loaded on the [WKND site](#) or an AEM site with the Adobe Data Layer enabled.

## Inspect the Button and Teaser schema

Before creating rules in the tag property, it is useful to review the [schema for the Button and Teaser](#) and inspect them in the data layer implementation.

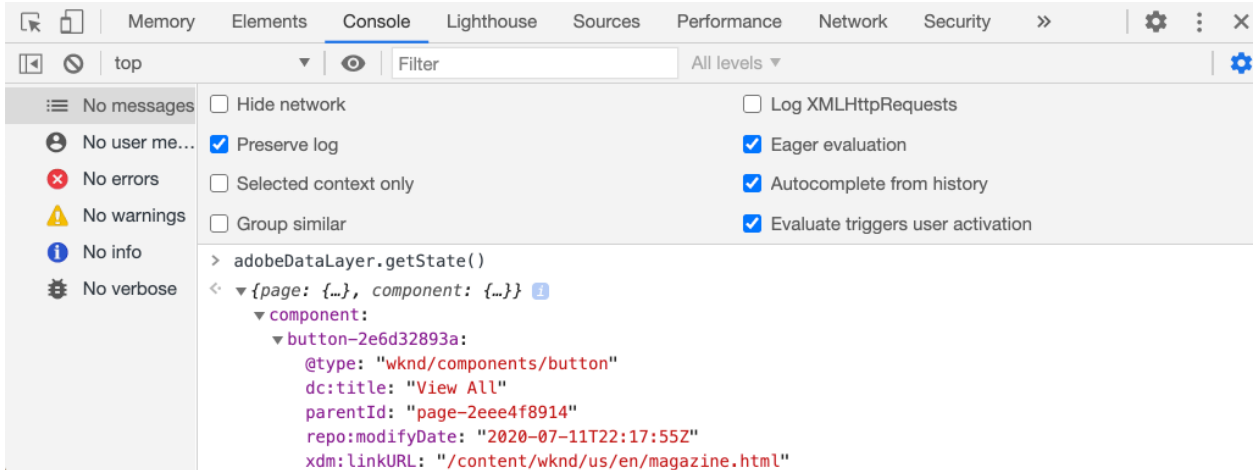
1. Navigate to [WKND Home page](#)
2. Open the browser's developer tools and navigate to the **Console**. Run the following command:

```
3. adobeDataLayer.getState();
```

Copy

Toggle Text Wrapping

Above code returns the current state of the Adobe Client Data Layer.



4. Expand the response and find entries prefixed with `button-` and `teaser-xyz-cta` entry. You should see a data schema like the following:

Button Schema:

```
button-2e6d32893a:
  @type: "wknd/components/button"
  dc:title: "View All"
  parentId: "page-2eee4f8914"
  repo:modifyDate: "2020-07-11T22:17:55Z"
  xdm:linkURL: "/content/wknd/us/en/magazine.html"
```

Copy

Toggle Text Wrapping

Teaser Schema:

```
teaser-da32481ec8-cta-adf3c09db9:
  @type: "wknd/components/teaser/cta"
  dc:title: "Surf's Up"
  parentId: "teaser-da32481ec8"
  xdm:linkURL: "/content/wknd/us/en/magazine/san-diego-surf.html"
```

Copy

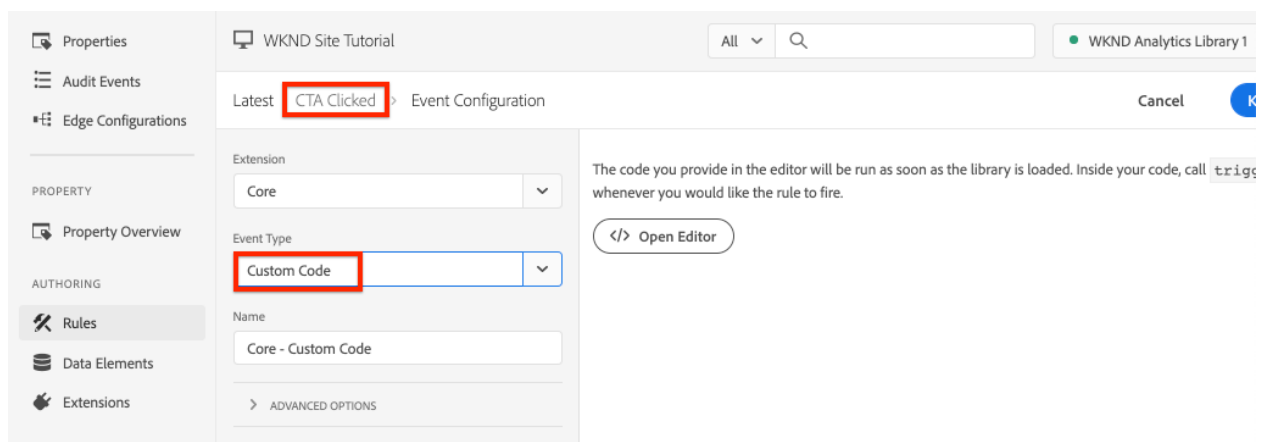
Toggle Text Wrapping

Above data details are based on the [Component/Container Item Schema](#). The new tag rule uses this schema.

## Create a CTA clicked rule

The Adobe Client Data Layer is an **event** driven data layer. Whenever any Core Component is clicked a `cmp:click` event is dispatched via the data layer. To listen for the `cmp:click` event, let's create a rule .

1. Navigate to Experience Platform and into the tag property integrated with the AEM Site.
2. Navigate to the **Rules** section in the Tag Property UI, then click **Add Rule**.
3. Name the rule **CTA Clicked**.
4. Click **Events** > **Add** to open the **Event Configuration** wizard.
5. For **Event Type** field, select **Custom Code**.



6. Click **Open Editor** in the main panel and enter the following code snippet:

```
7. var componentClickedHandler = function(evt) {
8.     // defensive coding to avoid a null pointer exception
9.     if(evt.hasOwnProperty("eventInfo") &&
    evt.eventInfo.hasOwnProperty("path")) {
10.         //trigger Tag Rule and pass event
11.         console.debug("cmp:click event: " + evt.eventInfo.path);
12.         var event = {
13.             //include the path of the component that triggered the event
14.             path: evt.eventInfo.path,
15.             //get the state of the component that triggered the event
16.             component: window.adobeDataLayer.getState(evt.eventInfo.path)
17.         };
18.
19.         //Trigger the Tag Rule, passing in the new `event` object
```

```
20.     // the `event` obj can now be referenced by the reserved name
    `event` by other Tag Property data elements
21.     // i.e `event.component['someKey']`
22.     trigger(event);
23. }
24.}
25.
26.//set the namespace to avoid a potential race condition
27.window.adobeDataLayer = window.adobeDataLayer || [];
28.//push the event listener for cmp:click into the data layer
29.window.adobeDataLayer.push(function (dl) {
30.    //add event listener for `cmp:click` and callback to the
    `componentClickedHandler` function
31.    dl.addEventListener("cmp:click", componentClickedHandler);
32.});
```

Copy

### Toggle Text Wrapping

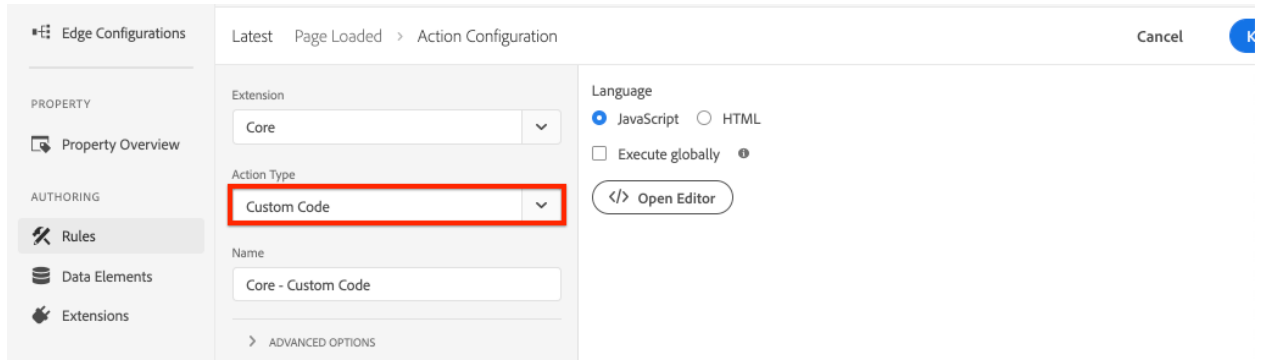
The above code snippet adds an event listener by [pushing a function](#) into the data layer. Whenever the `cmp:click` event is triggered the `componentClickedHandler` function is called. In this function, a few sanity checks are added and a new `event` object is constructed with the latest [state of the data layer](#) for the component that triggered the event.

Finally the `trigger(event)` function is called. The `trigger()` function is a reserved name in the tag property and it **triggers** the rule. The `event` object is passed as a parameter which in turn is exposed by another reserved name in the tag property. Data Elements in the tag property can now reference various properties using code snippet like `event.component['someKey']`.

33. Save the changes.

34. Next under **Actions** click **Add** to open the **Action Configuration** wizard.

35. For **Action Type** field, choose **Custom Code**.



36. Click **Open Editor** in the main panel and enter the following code snippet:

```
37. console.debug("Component Clicked");  
38. console.debug("Component Path: " + event.path);  
39. console.debug("Component type: " + event.component['@type']);  
40. console.debug("Component text: " + event.component['dc:title']);
```

Copy

Toggle Text Wrapping

The `event` object is passed from the `trigger()` method called in the custom event. The `component` object is the current state of the component derived from the data layer `getState()` method and is the element that triggered the click.

41. Save the changes and run a [build](#) in the tag property to promote the code to the [environment](#) used on your AEM Site.

## NOTE

It can be useful to use the [Adobe Experience Platform Debugger](#) to switch the embed code to a **Development** environment.

42. Navigate to the [WKND Site](#) and open the developer tools to view the console. Also, select the **Preserve log** checkbox.

43. Click one of the **Teaser** or **Button** CTA buttons to navigate to another page.

≡ **WKND**

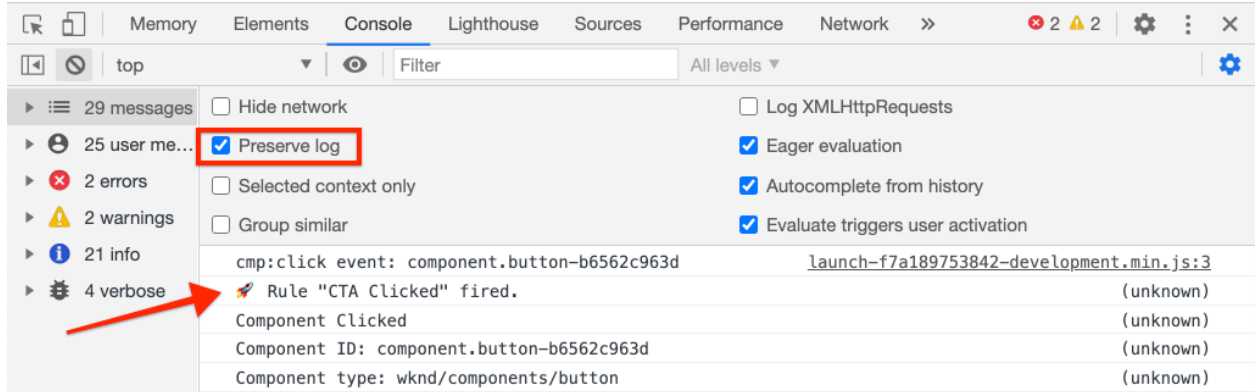
SEARCH

## WKND Adventures

Join us on one of our next adventures. Browse our list of curated experiences and sign up for one when you're ready to explore with us.

**VIEW TRIPS**

44. Observe in the developer console that the **CTA Clicked** rule has been fired:

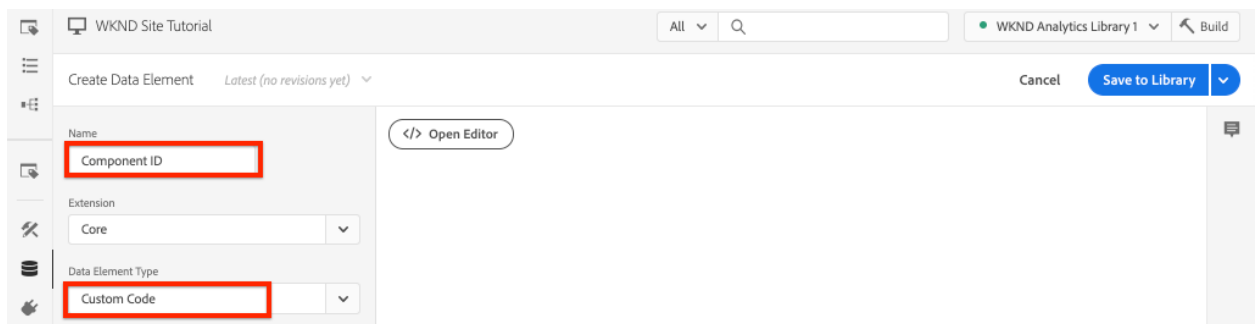


## Create Data Elements

Next create a Data Element to capture the component ID and title that was clicked. Recall in the previous exercise the output of `event.path` was something similar to `component.button-b6562c963d` and the value of `event.component['dc:title']` was something like “View Trips”.

## Component ID

1. Navigate to Experience Platform and into the tag property integrated with the AEM Site.
2. Navigate to the **Data Elements** section and click **Add New Data Element**.
3. For **Name** field, enter **Component ID**.
4. For **Data Element Type** field, select **Custom Code**.



5. Click **Open Editor** button and enter the following in the custom code editor:

```
6. if(event && event.path && event.path.includes('.')) {  
7.     // split on the `.` to return just the component ID  
8.     return event.path.split('.')[1];  
9. }
```

Copy

Toggle Text Wrapping

10. Save the changes.

## NOTE

Recall that the `event` object is made available and scoped based on the event that triggered the **Rule** in tag property. The value of a Data Element is not set until the Data Element is *referenced* within a Rule. Therefore it is safe to use this Data Element inside a Rule like the **Page Loaded** rule created in the previous step *but* would not be safe to use in other contexts.

### Component Title

1. Navigate to the **Data Elements** section and click **Add New Data Element**.
2. For **Name** field, enter **Component Title**.
3. For **Data Element Type** field, select **Custom Code**.
4. Click **Open Editor** button and enter the following in the custom code editor:

```
5. if(event && event.component &&
   event.component.hasOwnProperty('dc:title')) {
6.     return event.component['dc:title'];
7. }
```

Copy

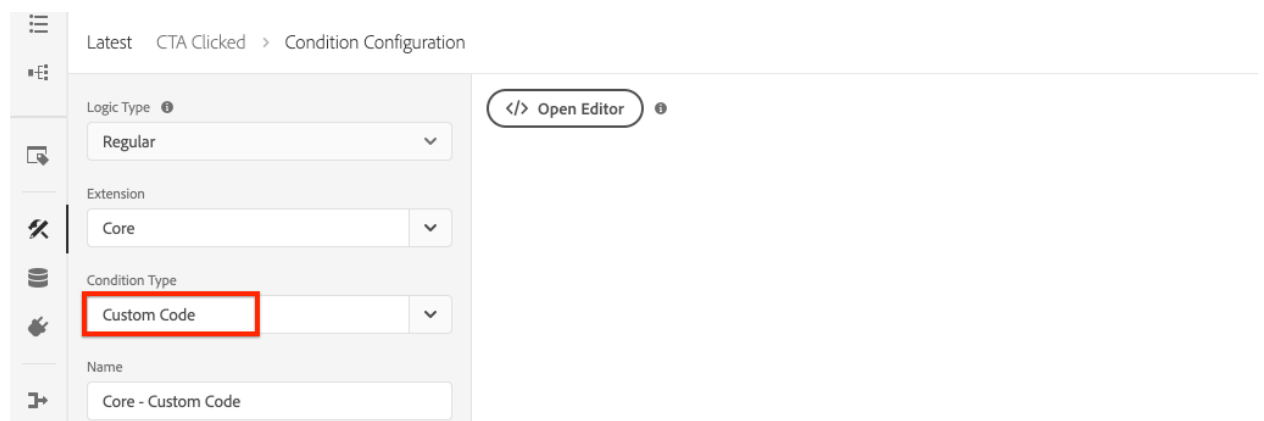
Toggle Text Wrapping

8. Save the changes.

### Add a condition to the CTA Clicked rule

Next, update the **CTA Clicked** rule to ensure that the rule only fires when the `cmp:click` event is fired for a **Teaser** or a **Button**. Since the Teaser's CTA is considered a separate object in the data layer, it is important to check the parent to verify it came from a Teaser.

1. In the Tag Property UI, navigate to the **CTA Clicked** rule created earlier.
2. Under **Conditions** click **Add** to open the **Condition Configuration** wizard.
3. For **Condition Type** field, select **Custom Code**.



4. Click **Open Editor** and enter the following in the custom code editor:

```
5. if(event && event.component && event.component.hasOwnProperty('@type'))
  {
6.     // console.log("Event Type: " + event.component['@type']);
7.     //Check for Button Type OR Teaser CTA type
8.     if(event.component['@type'] === 'wknd/components/button' ||
9.        event.component['@type'] === 'wknd/components/teaser/cta') {
10.        return true;
11.    }
12.}
13.
14.// none of the conditions are met, return false
15.return false;
```

Copy

Toggle Text Wrapping

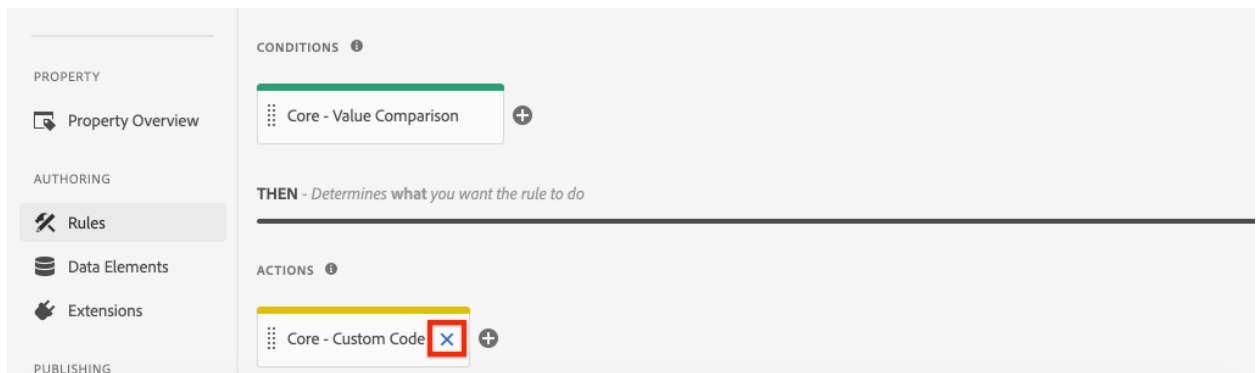
The above code first checks to see if the resource type was from a **Button** or if the resource type was from a CTA within a **Teaser**.

16. Save the changes.

Set Analytics Variables and trigger Track Link Beacon

Currently the **CTA Clicked** rule simply outputs a console statement. Next, use the data elements and the Analytics extension to set Analytics variables as an **action**. Let's also set an extra action to trigger the **Track Link** and send the collected data to Adobe Analytics.

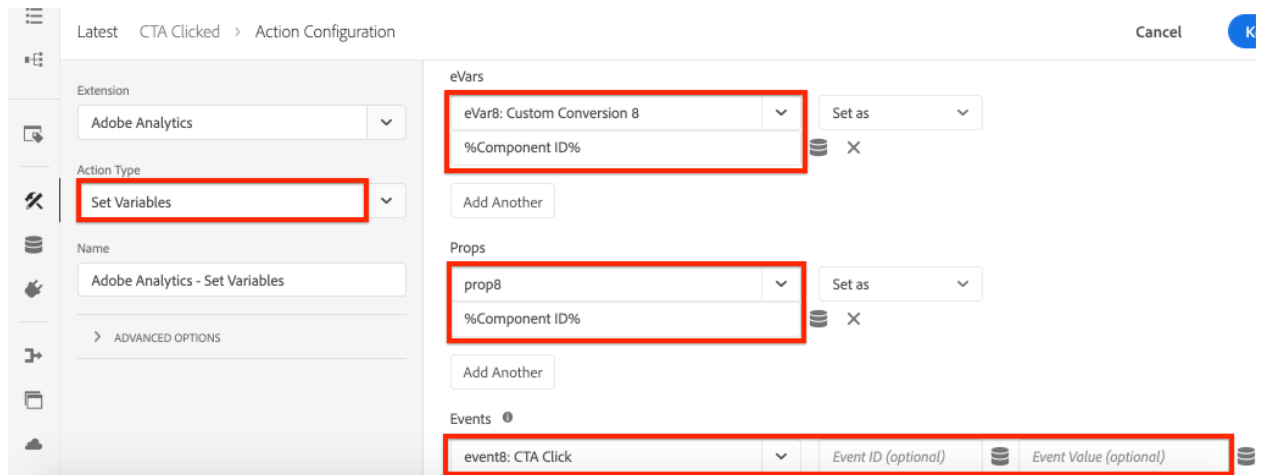
1. In the **CTA Clicked** rule, **remove** the **Core - Custom Code** action (the console statements):



2. Under Actions, click **Add** to create an action.



3. Set the **Extension** type to **Adobe Analytics** and set the **Action Type** to **Set Variables**.
4. Set the following values for **eVars**, **Props**, and **Events**:
  - evar8 - %Component ID%
  - prop8 - %Component ID%
  - event8



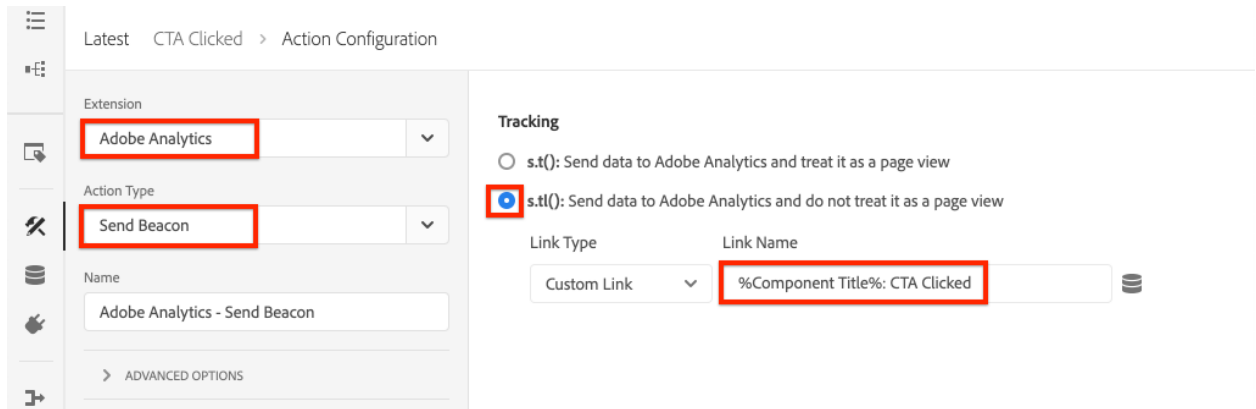
## NOTE

Here %Component ID% is used since it guarantees a unique identifier for the CTA that was clicked. A potential downside of using %Component ID% is that the Analytics report contains values like button-2e6d32893a. Using the %Component Title% would give a more human friendly name but the value might not be unique.

5. Next, add an extra Action to the right of the **Adobe Analytics - Set Variables** by tapping the **plus** icon:

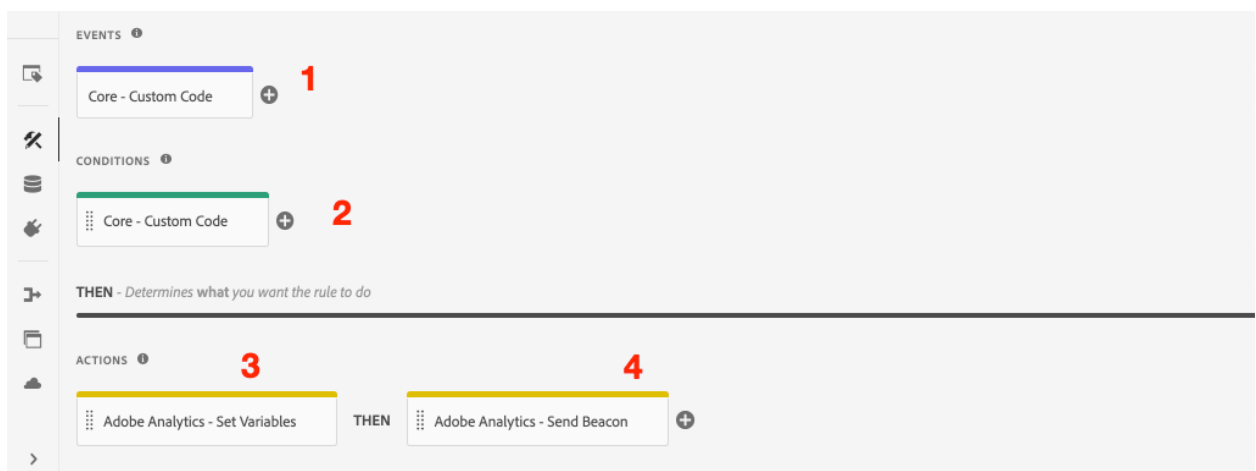


6. Set the **Extension** type to **Adobe Analytics** and set the **Action Type** to **Send Beacon**.
7. Under **Tracking** set the radio button to `s.tl()`.
8. For **Link Type** field, choose **Custom Link** and for **Link Name** set the value to: %Component Title%: CTA Clicked:



The above config combines the dynamic variable from the data element **Component Title** and the static string **CTA Clicked**.

9. Save the changes. The **CTA Clicked** rule should now have the following configuration:




- 1. Listen for the `cmp:click` event.
- 2. Check that the event was triggered by a **Button** or **Teaser**.
- 3. Set Analytics variables to track the **Component ID** as an **eVar**, **prop**, and an **event**.
- 4. Send the Analytics Track Link Beacon (and do **not** treat it as a page view).

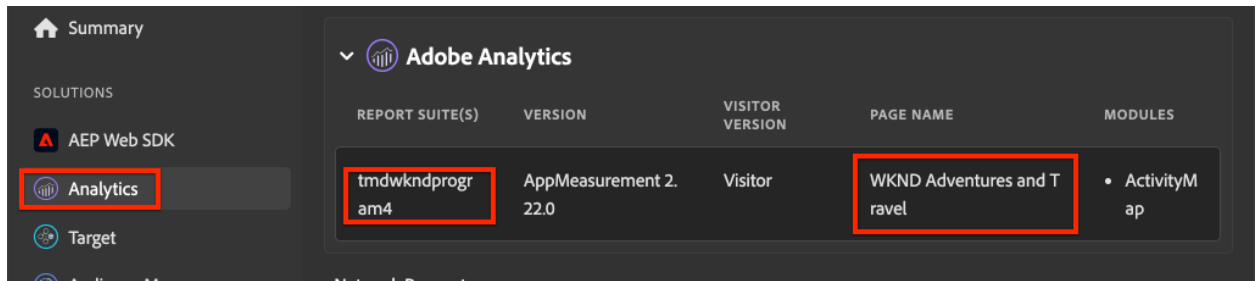
10. Save all the changes and build your tag library, promoting to the appropriate Environment.

Validate the Track Link Beacon and Analytics call

Now that the **CTA Clicked** rule sends the Analytics beacon, you should be able to see the Analytics tracking variables using the Experience Platform Debugger.

1. Open the [WKND Site](#) in your browser.

2. Click the Debugger icon  to open the Experience Platform Debugger.
3. Make sure that the Debugger is mapping the tag property to *your* Development environment, as described earlier and the **Console Logging** is checked.
4. Open the Analytics menu and verify that the report suite is set to *your* report suite.



The screenshot shows the Adobe Analytics interface. On the left, under 'SOLUTIONS', the 'Analytics' option is highlighted with a red box. The main area displays a table of report suites with the following data:

REPORT SUITE(S)	VERSION	VISITOR VERSION	PAGE NAME	MODULES
tmdwkndprogr am4	AppMeasurement 2. 22.0	Visitor	WKND Adventures and T ravel	• ActivityM ap

5. In the browser, click one of the **Teaser** or **Button** CTA buttons to navigate to another page.

≡ **WKND**

🔍 SEARCH

## WKND Adventures

Join us on one of our next adventures. Browse our list of curated experiences and sign up for one when you're ready to explore with us.

[VIEW TRIPS](#)

6. Return to the Experience Platform Debugger and scroll down and expand **Network Requests** > *Your Report Suite*. You should be able to find the **eVar**, **prop**, and **event** set.

Network Requests		
eVar 8	teaser-2122a9f81a-cta-6b28b693be	
eVar 9		Adventures
Events	event8	event4
Fallback Visitor ID	1EA1DDA4505FA6CB-2609FA3A468714FB	1EA1DDA4505FA6CB-2609FA3A468714FB
Java Enabled	N	N
Javascript Version	1.6	1.6
Link Name	View Trips: CTA Clicked	
Link Type	lnk_o	
Irt	211	
Page Name	WKND Adventures and Travel	Adventures
Page URL	https://wknd.site/us/en.html	https://wknd.site/us/en/adventures.html
pf	1	1
Prop 8	teaser-2122a9f81a-cta-6b28b693be	

- Return to the browser and open up the developer console. Navigate to the footer of the site and click one of the navigation links:

The screenshot shows the WKND MAGAZINE website with the 'ADVENTURES' link highlighted in the navigation menu. Below the website, the browser's developer console is open, displaying a message: "Condition "Custom Code" for rule "CTA Clicked" was not met." The message is highlighted with a red box. The console also shows various settings like "Hide network", "Log XMLHttpRequests", "Preserve log", "Eager evaluation", "Selected context only", "Autocomplete from history", "Group similar", and "Evaluate triggers user activation".

- Observe in the browser console the message *"Custom Code" for rule "CTA Clicked" was not met.*

The above message is because the Navigation component does trigger a `cmp:click` event *but* because of [Condition to the rule](#) that checks the resource type no action is taken.

## NOTE

If you don't see any console logs, ensure that **Console Logging** is checked under **Experience Platform Tags** in the Experience Platform Debugger.

Congratulations!

You just used the event-driven Adobe Client Data Layer and Tag in Experience Platform to track the clicks of specific components on an AEM site.

## **6.5 Adobe Analytics Implementation Debugging— Tools**

In the world of software development, debugging plays a very important role. It is a process of detecting and removing existing as well as potential errors. In the same way, when you are implementing Adobe Analytics, Analytics debugging refers to validating the payload that is being sent to the Adobe Analytics server.

Why is learning how to debug an Analytics beacon necessary? It is a must skill to learn as your implementation quality and speed improves. You don't need to wait for data to be populated in your Adobe Analytics report suite and then make the appropriate changes. This is a big help especially when you are working for a live project.

This means you check the variables and their values that your TMS is sending, even before Adobe Analytics processes it and stores it in the designated report suite. With the need defined, let me take you through a list of tools that are available to debug an Adobe Analytics implementation:

- Adobe Experience Platform Debugger
- Charles Web Debugging Proxy Application
- Debugger for Adobe Analytics Extension
- Browser Network filter

Adobe Experience Cloud Debugger

This is the official debugging tool provided by Adobe and is currently available as a [Chrome](#) and [Firefox](#) extension. The debugger is an all-encompassing toolset which means you can use it for validating all Adobe

Experience Cloud solutions including Analytics, AEP, Target, Audience Manager etc.

It has an array of features such as ability to inject or replace Adobe Launch embed code, federated login to auto populate property and the environment variables, lock mechanism to stay connected to the tab in focus, console logging, inbuilt Auditor etc. These rich and helpful features make it a must have tool for **debugging any implementation** project. And since it is supported by Adobe, you can expect feature upgrades which will make it more relevant and useful for your implementation needs.

Level of use: Easy

Charles Web Debugging Proxy Application

[Charles](#) app is not just a debugging tool. It is a web proxy that runs on your system. This makes it the middle layer to record and display all data that is sent and received from your browser. It means it can read all type of payloads including ones sent by Adobe Experience Cloud solutions.

The tool used to be my default go to utility until Adobe introduced the Experience Platform Debugger. Besides filtering for specific requests, it has some excellent features like Map Remote, Map Local, SSL Proxying etc. These were a great help for mapping development libraries on a production site to debug and find out errors for fixing issues. The tool has certain downsides too such as cost of accruing a license, a complicated setup for SSL tracking etc. But it is a great application that certainly should be explored.

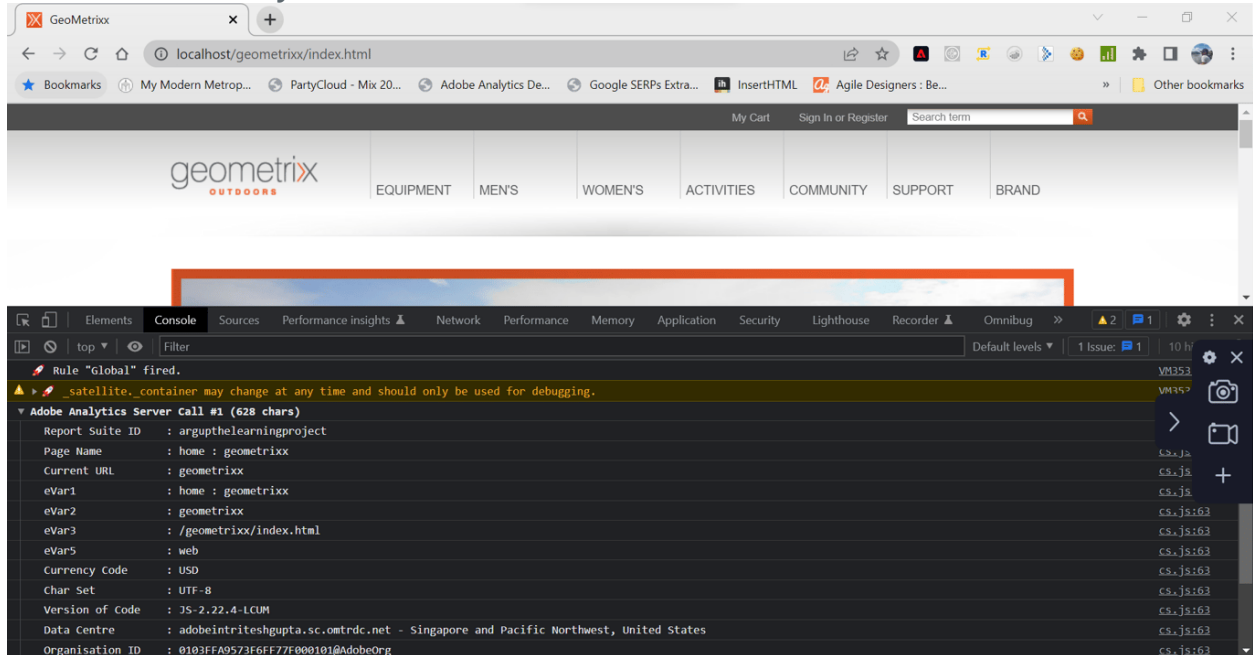
Level of use: Hard

Debugger for Adobe Analytics Extension

Sometimes we don't need a very elaborate setup like the one that Adobe Experience Platform debugger or Charles command. We want a quick view of the data that is sent to Adobe Analytics without moving away from the browser. Enter [Debugger for Adobe Analytics Chrome Extension](#). This is my favorite debugging tool as it allows me to view the variables within the Chrome window itself. I agree that the view is pretty basic, but it solves my

purpose 50% of the time. If you are looking at a tool to debug only Adobe Analytics payload, I highly advice you install it.

## Level of use: Easy

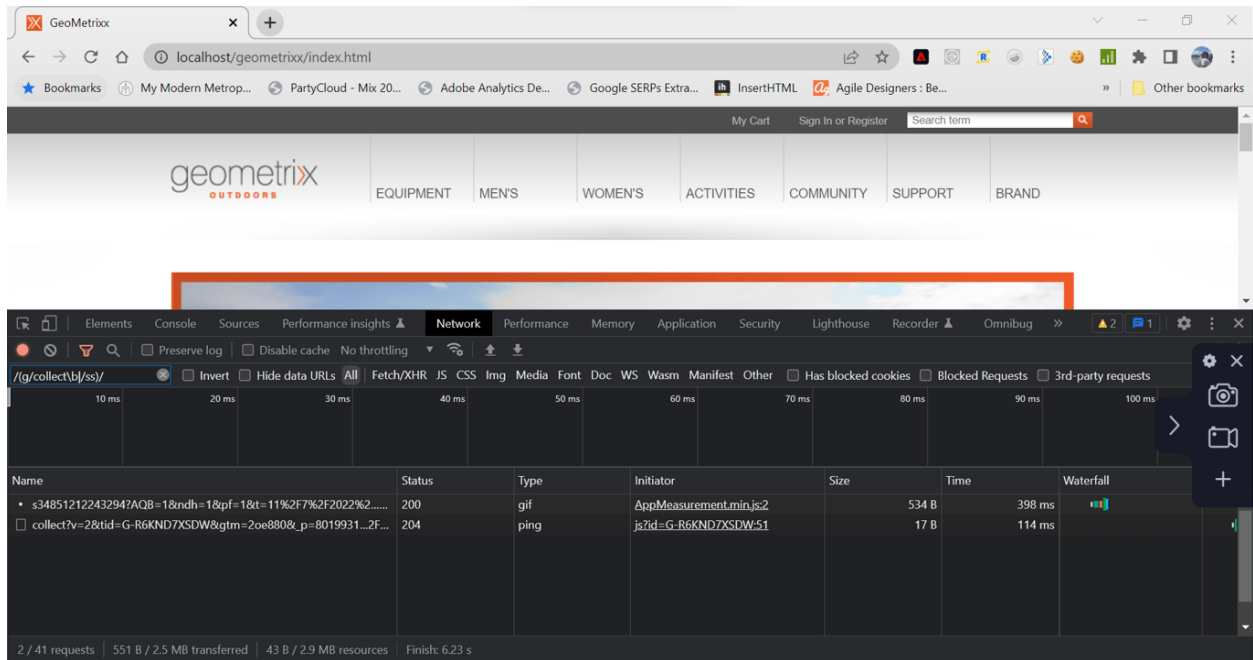


## Browser Network Window

This is another very basic application available for debugging purposes. This is again a good method to debug an Adobe Experience Cloud implementation. All you need to do is head over to the Network tab in the browser and filter the requests sent using certain solution specific commands like:

- Adobe Analytics : *b/ss*
- Adobe Target : *mbox*
- Google Analytics: *collect*

You can even write regular expression to combine two payloads such as:



`/(g\collect\b|ss)/` - For checking requests on Google And Adobe Analytics

`/(mbox\b|ss)/` - For checking requests on Adobe Analytics and Target

Level of use: Easy

That is all for this blog post. Do let me know what is your favorite go to tool for debugging an Adobe Analytics implementation. Drop me a line at [ritesh@thelearningproject.in](mailto:ritesh@thelearningproject.in).