



Adobe® Experience Cloud
Adobe Analytics Implementation

Contents

Implementation roadmap.....	9
Popular Implementation Links.....	10
Release Notes.....	10
Analytics Basics.....	11
Alerts.....	11
Analytics Code.....	11
Classifications.....	12
Conversion Variables (eVar).....	13
Data Collection.....	13
Code Manager.....	16
Data Layer.....	17
Events.....	17
Metrics.....	17
Processing Rules.....	18
Reports and Report Suites.....	19
Segments.....	20
Traffic Props and Conversion eVars.....	20
Comparing Props and eVars.....	21
Using Props as Counters.....	23
Counting Content Hierarchies.....	24
What is a Predefined Event?.....	24
What is a Custom Event?.....	26
Hash Collisions.....	26
Frequently Asked Questions about Analytics Implementation.....	28
Prepare to Implement Analytics.....	32
Business Requirements Document.....	32

Business Governance Questionnaire.....	32
Technical Questionnaire.....	32
Technical Specification.....	33
Review Website or Application.....	33
Dashboard Creation.....	35
Choose an implementation method.....	36
Getting Started with Analytics Implementation.....	36
Implement Analytics with Dynamic Tag Management.....	40
Create Web Property.....	40
Configure Hosting Options.....	42
Add Header and Footer Code.....	43
Verify Header and Footer Code.....	45
Add Adobe Analytics Tool.....	46
General.....	49
Library Management.....	50
Global Variables.....	52
Page Views and Content.....	53
Link Tracking.....	53
Referrers and Campaigns.....	54
Cookies.....	55
Customize Page Code.....	56
Frequently Asked Questions About the Adobe Analytics Tool.....	56
Create a Data Element.....	57
Manually implement Adobe Analytics (legacy).....	58
Create New Rule.....	60
Create Conditions for Event-Based Rules.....	61
Create Conditions for Page-Load Rules.....	63
Create Conditions for Direct-Call Rules.....	65
Set Up Actions for the Condition to Trigger.....	65
Test Unpublished Rules for Akamai Hosting.....	68
Test Rules for Library Download or FTP.....	68

Implement Analytics using JavaScript.....	69
Example Page Code and Global Configuration.....	70
About AppMeasurement for JavaScript.....	72
Migrating to AppMeasurement for JavaScript.....	73
AppMeasurement Plug-in Support.....	74
Accelerated Mobile Pages	75
Facebook Instant Articles.....	80
Additional Web and Mobile Measurement Libraries.....	83
Implementing Analytics using HTML image tags.....	84
Mobile Tracking over WAP and I-Mode Protocols.....	88
Variables for Analytics Implementation and Reporting.....	90
Configuration Variables.....	91
Context Data Variables.....	111
Dynamic Variables.....	112
Page Variables.....	115
Additional eVars and Events.....	151
Variables and Limitations.....	152
Illegal JavaScript Characters.....	156
Merchandising Variables.....	156
Implementing a Merchandising Variable.....	157
Instances on Merchandising Variables.....	159
The s.t() Function - Page Tracking.....	160
The s.tl() Function - Link Tracking.....	161
Automatic Tracking of Exit Links and File Downloads.....	163
Manual Link Tracking Using Custom Link Code.....	164
Link Tracking Using an Image Request.....	165
Setting Additional Variables for File Downloads, Exit Links, and Custom Links.....	165
Using Function Calls with Custom Link Code.....	166
Popup Windows with useForcedLinkTracking	167
Links from URL Shorteners.....	168
Link Tracking Variables in doPlugins.....	168
Validating File Downloads, Exit Links, and Custom Links.....	169
The s.sa() Function.....	169

The s.clearVars() Function.....	169
The s_gi() Function.....	170
Util.cookieRead.....	170
Util.cookieWrite.....	171
Util.getQueryParam.....	171
Offline Tracking.....	172
Data Collection.....	173
Data Collection Query Parameters.....	173
Data Collection HTTP Headers.....	178
Variable Overrides.....	179
Report Suite IDs - Dynamic Accounts.....	180
Collecting Data From Form Elements.....	182
Tracking Across Different Implementation Types.....	183
AJAX-Tracking Rich Media Applications.....	184
External Email Tracking.....	187
Implementing Adobe Opt-Outs.....	189
Implementation Plug-ins.....	192
Calling Plug-ins with doPlugins Function.....	192
s.abort flag.....	193
appendList.....	193
doPlugins Function.....	196
getAndPersistValue.....	197
getDaysSinceLastVisit.....	198
getLoadTime.....	199
getNewRepeat.....	200
getPageVisibility.....	201
getVisitStart.....	203
getPercentPageViewed.....	204
getPreviousValue.....	206
getQueryParam.....	207
getTimeParting.....	209
getValOnce.....	211
getVisitNum.....	213
hitGovernor.....	214

Internal Traffic.....	215
performanceTiming.....	216
trackTNT.....	219
Pathing.....	219
Enabling Pathing on a prop.....	220
Pathing by Campaign or Tracking Code.....	220
Reasons Pathing may not be Recorded.....	221
Moving from Section to Section.....	221
Moving from Page Template to Page Template.....	221
Segmenting Paths by User Type.....	221
Purchase Events.....	222
Event Serialization.....	223
Methods of Event Serialization.....	223
Identifying Unique Visitors.....	225
Custom Visitor ID.....	226
Experience Cloud ID Service.....	227
Analytics Visitor ID.....	227
Fallback ID Methods.....	227
Identifying Mobile Devices.....	228
Cross-Device Visitor Identification.....	230
Connecting Users Across Devices.....	230
Data Impact of Cross-Device Visitor Identification.....	231
Example visit.....	232
Visitors.....	233
Visits.....	234
Create Segments.....	234
Geo-Segmentation Data.....	234
Attribution and Persistence.....	234
Visitor Migration.....	236
Using Timestamps Optional.....	237
Redirects and Aliases.....	241
Analytics and Redirects.....	241
Implementing Redirects.....	242

Testing and Validation.....	245
Testing and Validation Process.....	245
Identifying the s_account Variable in the DigitalPulse Debugger.....	245
JavaScript JS File.....	245
Code Modifications.....	246
Variables and Values.....	246
Custom Variables.....	247
Implementation Acceptance.....	247
Data Accuracy Validation.....	248
Experience Cloud Debugger.....	249
Install the debugger in Chrome.....	249
Install the debugger in Firefox (not supported).....	250
Install the debugger in Internet Explorer (not supported).....	250
Packet Analyzers.....	250
NS_Binding_Aborted in Packet Monitors.....	251
Common Syntax Mistakes.....	251
Putting Analytics Code in the Head Tag.....	251
Using s.linkTrackVars and s.linkTrackEvents.....	253
Common Mistakes in the Products Variable.....	253
Setting the PageType Variable Correctly.....	255
Using White Space in Variable Values.....	255
Using Quotes.....	256
Replacing Your Analytics Code.....	256
Table of Common Syntax Mistakes.....	257
Vulnerability Scanner.....	257
Optimizing your Implementation.....	258
Page Naming.....	258
Variable Length.....	260
HTML Code Snippet.....	260
Javascript Library File.....	260
Caching Directives.....	260
Tables.....	261
File Compression.....	261

Secure Pages.....	261
Content Delivery Services/Networks.....	261
JavaScript File Location and Concurrency.....	261
Peering.....	262
Report to Variable Mapping.....	262
Variable to Report Mapping.....	268
Analytics for Digital Assistants.....	272
Digital Experience Architecture Overview.....	272
Where to implement Analytics.....	273
Analytics Implementation for Digital Assistants.....	273
Analytics Reporting for Digital Assistants.....	276
Example.....	276
Contact and Legal Information.....	279

Implementation roadmap

New Users

If you are new to Adobe Analytics, you can quickly create your first Analytics report suite (data repository) using the [Getting Started with Adobe Analytics](#) guide. Then, you can deploy Analytics code using [Launch](#).

Implementation Roadmap

Step	Task	Description
1	Choose an implementation method.	<p>Common ways to implement Analytics include:</p> <ul style="list-style-type: none"> • Launch (Recommended) <p>This guide tells you everything you need to know about using Adobe's website tag and mobile SDK management capabilities and how to implement them.</p> <ul style="list-style-type: none"> • Dynamic Tag Management <p>This guide contains Analytics-specific information to guide you through a Dynamic Tag Management implementation.</p> <ul style="list-style-type: none"> • JavaScript <p>This guide contains a description of data collection variables and details on implementing data collection code in JavaScript, including video.</p> <ul style="list-style-type: none"> • Analytics SDKs <p>Use Analytics SDKs to manage:</p> <ul style="list-style-type: none"> • Mobile apps on iOS • Mobile apps on Android
2	Set up the Experience Cloud ID service.	(Formerly <i>Visitor ID service</i> .) See Set Up the Experience Cloud ID Service for Analytics .
3	Use the chosen implementation method to update and deploy page code.	<p>Place the page code just after the opening <code><body></code> tag on each page you want to track. At a minimum, update the following variables:</p> <ul style="list-style-type: none"> • <code>var s=s_gi("INSERT-RSID-HERE")</code> • <code>s.pageName="INSERT-NAME-HERE" // for example, s.pageName=document.title</code>
4	Validate the implementation.	Testing and Validation Provides information about validating your implementation.
5	Use the Adobe Experience Cloud debugger to verify that data is being sent.	Install the Experience Cloud Debugger . After it is installed, load a page where you have deployed page code and then open the debugger. The debugger displays details about the collection data that was sent.

More Information

For information about the differences between Launch, Dynamic Tag Management and JavaScript methods, see [Choose an implementation method](#).

For a concise overview of the getting started process and help on quickly setting up your first Analytics report suite, see [Getting Started with Analytics Implementation](#) in the Getting Started with Analytics guide.

Popular Implementation Links

During implementation, you might find the following documents useful:

- [Accelerated Mobile Pages](#)
- [Experience Cloud Debugger](#)
- [Implementation Plug-ins](#)
- [Variables for Analytics Implementation and Reporting](#)
- [Processing Rules](#)
- [VISTA rules](#) (in Admin Help)
- [Experience Cloud ID Service](#) documentation
- [Experience Cloud & Core Services](#) documentation

Release Notes

Release notes can be found at the following locations:

- [Experience Cloud - All Solutions](#)
- [AppMeasurement Libraries - All Platforms](#)

Analytics Basics

Learn about the basics of Analytics, including terms, what's in the code, and how data is collected.

Logging In to Adobe Analytics

To learn how to log in to Analytics, refer to [Logging In to Reports and Analytics](#) in the Reports & Analytics Interface Help.

Experience Cloud Debugger

The Adobe Debugger (previously DigitalPulse Debugger) is a free tool provided by Adobe that lets you view the data being collected from your site on any given page.

See .

Alerts

Intelligent Alerts let you create and manage alerts in Analysis Workspace, complete with alert preview and rule contribution.

You can

- Build alerts based on anomalies (90%, 95%, or 99% thresholds; % change; above/below).
- Preview how often an alert will trigger.
- Send alerts by e-mail or SMS with links to auto-generated Analysis Workspace projects.
- Create "stacked" alerts that capture multiple metrics in a single alert.

You can access this new Alerts system from **More > Alerts** in any report in Reports & Analytics.

See [Alerts](#) in the Reports and Analytics Help.

Analytics Code

Data is sent to a report suite to display in reporting. The easiest and most common way to send data to Analytics is by using the DTM implementation. You can also enter the code with the JavaScript implementation.

For information about using Dynamic Tag Management to create the header and footer code for you, see [Add Header and Footer Code](#).

Here is an example of AppMeasurement JavaScript file, showing the code parts:

```

s = new AppMeasurement();
s.account="atsgeometrixoutdoors"

```

Report Suite ID

```

/***** CONFIG SECTION *****/
/* You may add or alter any code config here. */
s.charset="UTF-8"
/* Conversion Config */
s.currencyCode="USD"
/* Link Tracking Config */
s.trackDownloadLinks=true
s.trackExternalLinks=true
s.trackInlineStats=true
s.linkDownloadFileTypes="exe,zip,wav,mp3,mov,mpg,avi,wmv,pdf,doc,docx,xls,xlsx,ppt,pptx"
s.linkInternalFilters="javascript:ats.adobe.com"
s.linkLeaveQueryString=false
s.linkTrackVars="None"
s.linkTrackEvents="None"
/* DST areas in US and Canada (no DST in AZ or SK, varies by county in IN) */
s.dstStart="3/10/2013";
s.dstEnd="11/3/2013";
s.currentYear="2013";

/* WARNING: Changing any of the below variables will cause drastic
changes to how your visitor data is collected. Changes should only be
made when instructed to do so by your account manager.*/
s.visitorNamespace="geometrixoutdoors"
s.trackingServer="geometrixoutdoors.d1.sc.omtrdc.net"

```

Configuration Section

```

s.usePlugins=true
s.doPlugins = function(s) {
  s.campaign = s.Util.getQueryParam("cid");
  s.prop17=s.getTimeParting('h','-7'); // Set hour
  s.prop18=s.getTimeParting('d','-7'); // Set day
  s.prop19=s.getTimeParting('w','-7'); // Set weekday
};

/*
 * Plugin: getTimeParting 2.0 - Set timeparting values based on time zone
 */
s.getTimeParting=new Function("t","z","",
+"var s=this,cy;dc=new Date('1/1/2000');"
+"if(dc.getDay()!6||dc.getMonth()!0){return'Data Not Available'}"
+"else{z=parseFloat(z);var dsts=new Date(s.dstStart);"
+"var dste=new Date(s.dstEnd);fl=dste;cd=new Date();if(cd>dsts&&cd<fl)"
+"{z=z+1}else{z=z};utc=cd.getTime()+(cd.getTimezoneOffset()*60000);"
+"tz=new Date(utc + (3600000*z));thisy=tz.getFullYear();"
+"var days=['Sunday','Monday','Tuesday','Wednesday','Thursday','Friday',"
+"Saturday'];if(thisy!=s.currentYear){return'Data Not Available'}else{"
+"thish=tz.getHours();thismin=tz.getMinutes();thisd=tz.getDay();"
+"var dow=days[thisd];var ap='AM';var dt='Weekday';var mint='00';"
+"if(thismin>30){mint='30'}if(thish>=12){ap='PM';thish=thish-12};"
+"if (thish==0){thish=12};if(thisd==6||thisd==0){dt='Weekend'};"
+"var timestring=thish+":"+mint+ap;if(t=='h'){return timestring}"
+"if(t=='d'){return dow};if(t=='w'){return dt}};");

/*
===== DO NOT ALTER ANYTHING BELOW THIS LINE ! =====
AppMeasurement for JavaScript version: 1.0.2
Copyright 1996-2013 Adobe, Inc. All Rights Reserved
More info available at http://www.omniture.com
*/
function AppMeasurement(){var s=this;s.version="1.0.2";var w=window;if(!
w.s_c_in)w.s_c_in=[],w.s_c_in=0;s._il=w.s_c_in;s._in=w.s_c_in;s._il[s._in]=s;w.s_c_in+

```

Plug-ins Section

**AppMeasurement for
JavaScript Base Code**

Classifications

Classifications are created by grouping (classifying) granular data from a source report.

For example, you might want to analyze display ads, but they are mixed with email, partner, text ad, and social media campaigns. You can create groups so that you can analyze them separately.

For more details, see:

- [Classifications](#) in the Analytics Help and Reference guide

Conversion Variables (eVar)

The Custom Insight Conversion Variable (or eVar) is placed in the Adobe code on selected web pages of your site. Its primary purpose is to segment conversion success metrics in custom marketing reports.

eVars are best used to measure cause and effect, such as:

- Which internal campaigns influenced revenue
- Which banner ads ultimately resulted in a registration
- The number of times an internal search was used before making an order



Important: When implementing Analytics, it is important to know which eVars you will use, and how many. You should also understand how to configure those eVars in the Admin Console. For detailed information about eVars, see [Conversion Variables \(eVar\)](#) in the Analytics Help and reference documentation.

An eVar can be visit-based and function similarly to cookies. Values passed into eVar variables follow the user for a predetermined period of time.

When an eVar is set to a value for a visitor, Adobe automatically remembers that value until it expires. Any success events that a visitor encounters while the eVar value is active are counted toward the eVar value.



Note: Only a single value can be stored in an eVar in an image request. If multiple values are desired in an eVar value, we recommend that you implement [List variables \(list vars\)](#).

For more information about variables, see:

- [Variables for Analytics Implementation and Reporting](#) in this help
- [Variables - How They Are Used in Reporting](#) in the Analytics help
- [Page Variables](#) in this help
- [Campaign variable](#) in this help
- [Products Variable](#) in this help
- [Products Variable](#) in the Mobile SDK documentation

Data Collection

Learn how visits to your web site become a report in Adobe Analytics.

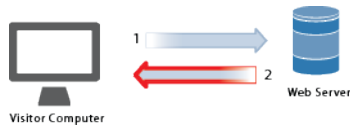
Adobe has created multiple ways to send data into Analytics. These methods include tracking information in real-time from:

- Applications that can access the Internet
- Campaigns
- Client-server applications
- Emails
- Mobile devices
- Web-based kiosks
- Web sites

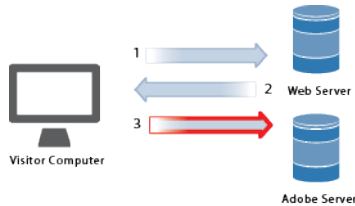
1. When a visitor comes to your site, a request is made to your web server.



2. Your site's web server sends the page code information, and the page displays in the browser.



3. The page loads, and the Analytics JavaScript code runs.



The JavaScript code sends an image request to the Adobe server, passing the variables, metrics, and page data that you defined in your implementation.

Example JavaScript Code: The JavaScript code is placed within the body tags of a web page:

The image shows a screenshot of the Geometrix website with a navigation menu (WOMEN, MEN, JEWELRY) and a search bar. A large, semi-transparent box with a torn-edge effect is overlaid on the page, containing JavaScript code. The code includes traffic and conversion variables, and an image request function.

```

<script language="JavaScript" type="text/javascript"
scr="http://www.geometrix.com/AppMeasurement.js"></script>
<script language="JavaScript" type="text/javascript"><!--

/* Traffic Variables */
s.pageName="Women:Fine Apparel:Red Dress"
s.channel="Women"
s.server="Geometrix UK"
s.pageType=""
s.prop1="Fine Apparel"
s.prop2=""
s.prop3=""
s.prop4=""
s.prop5=""

/* Conversion Variables */
s.campaign=""
s.state=""
s.zip=""
s.events="prodView"
s.products=";101341"
s.purchaseID=""
s.eVar1=""
s.eVar2=""
s.eVar3=""
s.eVar4=""
s.eVar5=""


var s_code=s.t();if(s_code)document.write(s_code)//--></script>

```

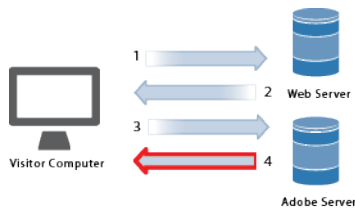
Example Image Request: A snippet of an image request with the page name outlined:

```

okieDomainPeriods=2&pageName=adobe.com%3Aproducts%3Aphotoshop&g=https%3A%2F
S&r=https%3A%2F%2Fwww.adobe.com%2Fsearch=Products&server=www.adobe.com&v0=D
4%3A22%3A0%2C70363%3A2%3A0%2C&l1=seg
3Adobe.com%3Aproducts%3Aphotoshop&v5=KLXLS&c12=adobe.com&c13=100&v16=D
/products
=1009&p=Shockwave%20Flash%3BAdobeAAMDetect%3BJava%20Applet%20Plug-in%3BDefault
%20Plug-in%207.7.3%3BGoogle%20Talk%20Plugin%20Video%20Renderer%3BGoogle%20Talk
le%20Earth%20Plug-in%3BCitrix%20Receiver%20Plug-in%3BAdobeExManDetect%3BFlip4Mac
    
```

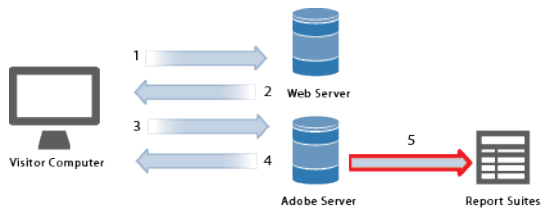
 **Note:** Each image request contains a random number string to prevent browser caching and ensure that subsequent image requests are made by the browser.

4. Adobe returns a transparent pixel image.



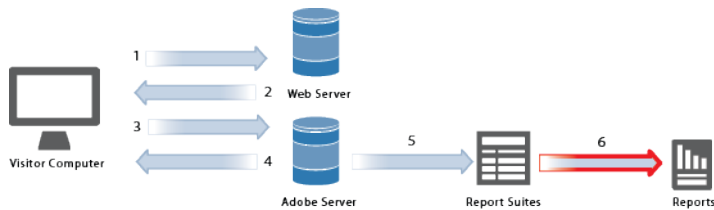
The code automatically collects additional details (such as operating system, browser type, browser height and width, IP address, and browser language).

5. Adobe servers store web analysis data in *report suites* (your data repository).

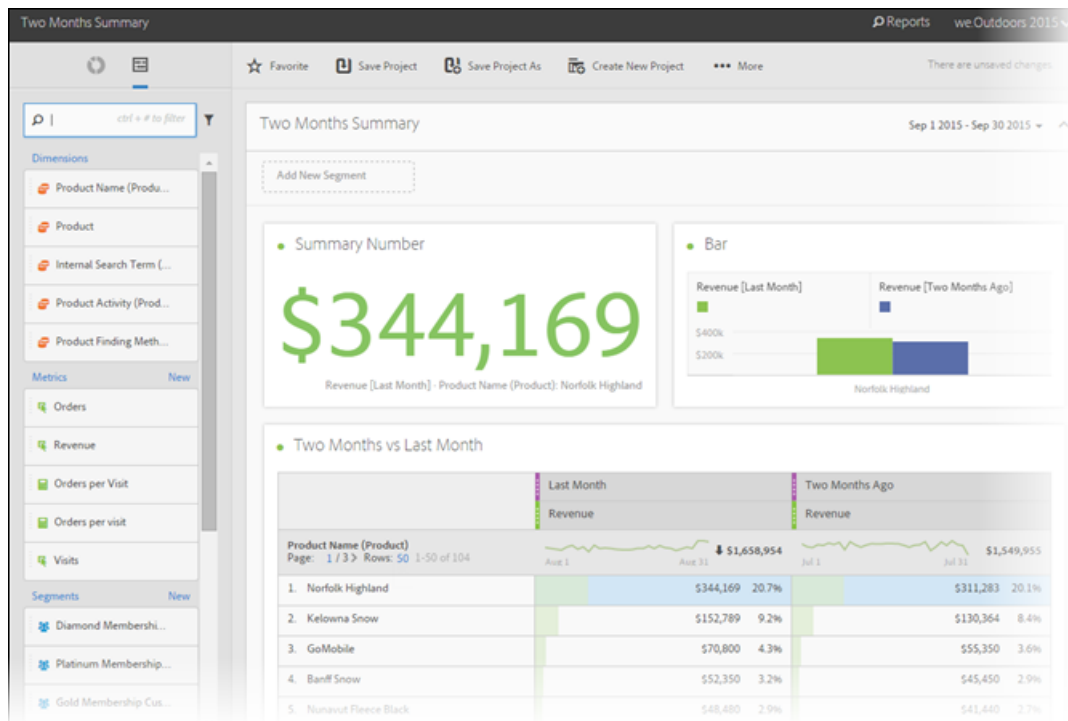


A *report suite* defines the complete, independent reporting on a chosen website, set of websites, or subset of web pages.

6. Report suite data populates the reports that you can access in a web browser.



Example report:



The JavaScript code execution occurs quickly and does not noticeably affect page load times. This approach allows you to count pages that were displayed when a visitor clicked **Reload** or **Back** to reach a page, because the JavaScript runs even when the page is retrieved from cache.

For more details, see:

- [Data Collection](#) in this help documentation
- [Create a Data Element](#) in this help documentation
- [Data Warehouse](#) in the Analytics Help and Reference guide
- [Ad Hoc Analysis](#) help
- [Exclude by IP Address](#) in the Analytics Help and Reference guide
- [Data Sources](#) whitepaper
- [Data Connectors](#) in Using FTP and sFTP with the Adobe Experience Cloud
- [Clickstream Data Feeds help](#)

Code Manager

Code manager lets you download data collection code for web and mobile platforms.

Analytics > Admin > Code Manager.

After you download the library, you must configure the code to send data to the correct tracking server and report suite. Additional implementation resources are available at [Developer & Implementation](#).

Code Manager Page Descriptions

Column	Description
Name	The name matches the platform where you want to enable data collection. Native libraries are provided for each platform listed in this column.
Type	Type of applications that can be measured using each library.
Version	List the latest version of the library. Click the version number to view the release history.
Documentation	View the library reference documentation.

Data Layer

A "data layer" is a framework of JavaScript objects that developers insert into pages. The data layers can be used by tracking tools (including tag management systems like Dynamic Tag Management) to populate reports.

Implementing a data layer on your site provides ultimate control and flexibility over your implementation and allows easiest maintenance in future phases. The names of these JavaScript objects are theoretically arbitrary, but the best practice is to use something consistent and predictable. The developers may already have a data layer, or a preference for the format. There are few different standards the tracking community has created as a starting point. The [Data Layer for Analytics Implementation](#) specification document uses the W3C standard "digitalData" object that is accepted by the widest variety of tracking technology, in case there is a need to use the data layer for more than this DTM implementation.

Events

Events keep track of when a visitor has performed a specified action.

An event should be considered a milestone within a site. Success events are most commonly populated on the final confirmation page of a process, such as a registration process or newsletter sign-up.

Common types of events include:

- Success events
- Currency events
- Custom events

For detailed information, refer to [Configure Events](#).

Success events are described in detail in the [Analytics Help and Reference](#).

Metrics

[Metrics](#) are quantitative information about visitor activity, such as Views, Click-Throughs, Reloads, Average Time spent, Units, Orders, and Revenue.

Metrics are the foundation of reports and help you view and understand data relationships. They let you perform side-by-side comparisons of different data sets about your website.

Metric Type	Definitions
Traffic metrics	<p>Page View: One Web page load in a user's browser (one execution of the Reports & Analytics code).</p> <p>Visit: Any number of page views when visitor comes to your site. A visit ends after 30 minutes of inactivity.</p> <p>Unique Visitor: A person visiting your site for the first time during a given time frame, such as Hour, Day, Week, Month, Quarter or year. (This also includes Unique Visitors for any time frame.)</p>
Conversion metrics	These show data about success events, such as purchases, downloads, or any other action that you want users to take on your website.
Video metrics	Analytics provides support for tracking a number of video metrics, including total views, time spent, and completion rates.
Social metrics	You can measure your brand's presence on the social web. Social metrics work with Analytics standard metrics. By combining these with calculated metrics, you can view a report that shows how often a product is mentioned, gauge product sentiment, and see how Social metrics correlate with Analytics key performance indicators.
Calculated metrics	<i>Calculated metrics</i> enable you to combine metrics to create mathematical operations that are used as new metrics. These metrics can be created for a report to which you add metrics. Administrators can create calculated metrics for all users of a report suite.

See [Metrics](#) in Analytics Help and Reference for metric definitions.

Processing Rules

Processing rules simplify data collection and manage content as it is sent to reports.

Processing rules help simplify interaction with IT groups and Web developers by providing an interface to:

- Set an event on the product overview page
- Populate campaign with a query string parameter
- Concatenate category and page name in a prop for easier reporting
- Copy an eVar into a prop to see paths
- Clean up misspelled site sections
- Pull internal search terms or a campaign ID from the query string into an eVar

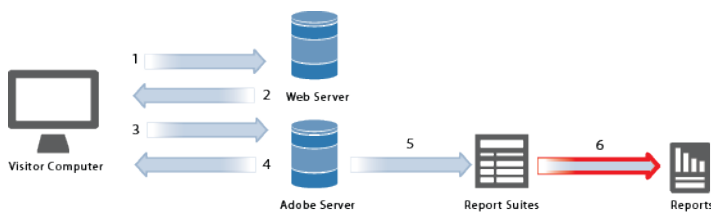
For details about processing rules, see:

- [Processing Rules](#) in the Analytics Help and Reference guide

- [Create New Rule](#) in this help documentation
- [Getting Started with Marketing Channels](#) in the Marketing Channels documentation
- [VISTA Rules](#) in the Analytics Help and Reference guide
- [Bot Rules](#) in the Analytics Help and Reference guide

Reports and Report Suites

A [report suite](#) defines the complete, independent reporting on a chosen website, set of websites, or subset of web pages. Usually, a report suite is one website, but it can be a global segment where you have combined several sites' numbers to get totals. When you log in to the marketing [reports](#), [ad hoc analysis](#), and [report builder](#), you select one report suite to use (except when you use roll-ups that combine report suites).



Reports provide information about the data collected by Analytics, based on specific parameters.

You can run an *Analytics report* after implementing Adobe Analytics. Reporting provides insights into your traditional web-based channels as well as evolving channels like mobile, video, and social networking. Some examples of marketing reports include:

- How many people visit your site
- How many of those visitors are unique visitors (counted only once)
- How they came to the site (such as whether they followed a link or came there directly)
- What keywords visitors used to search site content
- How long visitors stayed on a given page or on the entire site
- What links visitors clicked, and when they left the site
- Which marketing channels are most effective at generating revenue or conversion events
- How much time they spent watching a video
- Which browsers and devices they used to visit your site

High-level report types include:

- **Traffic:** Gives you in-depth insight into how visitors interact with your website, and your customized traffic statistics.
- **Conversion:** Displays information about success indicators that you define.
- **Paths:** Enable you to track and record entire browsing paths of visitors.

You can use [Analysis Workspace](#) to remove the typical limitations of a single Analytics report. It provides a robust, flexible canvas for building custom analysis projects. Drag-and-drop any number of data tables, visualizations, and components (dimensions, metrics, segments, and time granularities) to a project. Instantly create breakdowns and segments, create cohorts for analysis, create alerts, create segments, and curate reports for sharing with anyone in your business.

See Also

- [Reports and Analytics](#) help
- [Real-Time Reports](#)

- [Adobe Report Builder](#) help
- [Data Extracts](#) in the Reports and Analytics help
- [ClickMap](#) in the Reports and Analytics help
- [Activity Map](#) in the Reports and Analytics help
- [Report Suite Manager](#)
- [Analytics Product Comparison and Requirements](#)
- [Reports and Analytics Interface Help](#)
- [Analysis Workspace Help](#)
- [Report Descriptions](#)
- [Dashboards and Reportlets](#)
- [Bookmarks](#)
- [Virtual Report Suites](#)
- [Anomaly Detection](#) in the Report Builder Documentation
- [Contribution Analysis](#) in the Contribution documentation

Segments

Segments are custom subsets of data, or data filtered by rules that you create.

Segments are based on:

- Hits
- Visitors
- Visits

For information about using segments for cross-device visitor identification, see [Create Segments](#).

For in-depth information on Adobe Analytics Segmentation, please refer to the [Segmentation Guide](#).

Traffic Props and Conversion eVars

Custom traffic variables, also called props (s.prop) or **property** variables, are counters that count the number of times each value is sent into Analytics.

When determining which variables are assigned where, it is important to understand the differences between Prop and eVar functionality. Understanding these differences allows your organization to decide which type of variable is best to use. For detailed information, see [Comparing Props and eVars](#).

Props also let you correlate custom data with specific traffic-related events. These variables are embedded in the Analytics code on each page of your website. Through **s.prop** variables, Analytics lets you create custom reports, unique to your organization, industry, and business objectives.

For example, if you are an automobile manufacturer, you may be interested in seeing "Most Popular Car Model" to complement your "Pages" report. You can accomplish this by allocating one of your traffic properties to represent car model. Then implement your code to pass in car model on the appropriate pages.

 **Note:** Analytics supports up to 75 **s.prop** variables.

Props are used in pathing reports or in correlation reports. For example, **property** variables can be used to show content type, sub-section, or template name. The resulting **Custom Traffic** reports show which content type, sub-section, or template is viewed most often.

There are endless business questions that can be answered through the custom traffic variables, depending on what you are capturing from your website. The following list contains a few common goals and objectives:

- Understanding user navigation through the website
- Understanding internal user search behavior
- Segmenting traffic by navigation or category
- Segmenting visitor behavior by demographics

eVars (or **Custom Conversion Insight** variables) are used to identify how well specific attributes or actions contribute to success events on your site. For example, for a media site, eVars may be used to identify how well internal promotions bring visitors to register. When a visitor clicks on the internal promotion, an eVar can be used to store a unique identifier for that promotion. When the same visitor completes registration and a custom success event is fired, the original unique identifier receives credit for the registration event.

In a conversion site, eVars may be used to track how logged-in visitors compare to non-logged in visitors in completing a purchase. When a visitor logs in, an eVar is set to "logged in." When that visitor reaches the checkout page, the checkout event is attributed to the "logged in" value. When a visitor reaches the Thank You page after purchasing, the products and purchase amounts are attributed to the "logged in" value. The resulting **Custom eVar** report shows the total number of checkouts and orders for "logged in" and "non-logged in" visitors.

For additional information, see [Traffic Variable](#) in the Analytics Help and Reference.

For information about setting up properties in Digital Tag Management, see [Create Web Property](#).

Comparing Props and eVars

There are several types of variables available in the Experience Cloud. The two most popular types, Props and eVars, allow your organization to report on custom dimensions to your site that standard out-of-the-box reports do not offer.

When determining which variables are assigned where, it is important to understand the differences between Prop and eVar functionality. Understanding these differences allows your organization to decide which type of variable is best to use.

Props vs eVars

The following are the main differences between props and eVars:

- **Naming convention:** Props are considered traffic variables, meaning they are used to report on popularity of various dimensions of your site. eVars are considered conversion variables. They are used to determine which dimensions of your site contribute the most to success events.
- **Persistence:** Props do not persist beyond the image request they were fired on. They cannot be associated with other variables that are not in the same image request. eVars, however, are persistent. A back-end variable is used to preserve the value originally fired so it can associate itself with success events later on.
- **Success events:** Success events, also known as conversion events, are metrics that measure the number of times a visitor reaches a goal. This event can be anything from purchasing something on your site, to subscribing to a newsletter. eVars are designed to report on conversion events, to show you which values are most successful in influencing visitors to reach your goals. Traffic variables do not have this same functionality. However, you can view participation metrics if you configure your report suite correctly.
- **Pathing:** Props can use pathing, which allows your organization to see a given path a user took within the context of the variable being viewed. An Adobe representative can enable pathing, if requested. eVars cannot use pathing.
- **Potentially available metrics:** The metrics available between props and eVars vary widely based on the variable's settings and data platform/version. The following list illustrates what can be enabled, not what is enabled by default. If you want a specific metric in reporting but do not see it, have one of your organization's supported users contact Customer Care.

Metric	Props (Traffic Variables)	eVars (Conversion Variables)
Average Page Depth	✓	
Average Time Spent	✓	
Bounce Rate	✓	✓
Bounces	✓	✓
Calculated Metrics	✓	✓
Custom Conversion Events		✓
Entries	✓	✓
Exits	✓	✓
Instances	✓	✓
Page Views	✓	✓
Participation Metrics	✓	✓
Purchase Metrics		✓
Reloads	✓	
Shopping Cart Metrics		✓
Single Access	✓	
Total Time Spent	✓	✓
Unique Visitors	✓	✓
Visits	✓	✓

- **Breakdowns:** Props use correlations, which display page views for other traffic variables fired in the same image request. eVars use subrelations, which provide a breakdown on other conversion variables in relation to success events.

Exclusive advantages to Props or eVars

With the release of version 15, the capabilities between Props and eVars are much less distinct. eVars have recently been updated to include capabilities such as visits/unique visitors with minimal processing load, as well as pathing metrics.

Props hold a couple advantages of eVars, some of which can be circumvented:

- Prop data is collected and available in reporting almost instantly. eVars can take upwards of 30 minutes to appear in report suite data.
- All Props can have flowchart-like reports enabled, which let you see the path visitors take to your site. These pathing flow reports are available for both Props and eVars in Ad Hoc Analysis.
- Props can be correlated multiple levels, where eVars can only be subrelated once. This limitation can be mitigated by using segmentation, giving identical data as correlations.
- Participation metrics allow you to see what prop values participated before a success event.

eVars on the other hand hold several advantages over the limited nature of Props:

- eVars can use success events as metrics. Props cannot.
- eVar expiration can be customized, including having an expiration of every hit (no persistent values whatsoever). Props don't persist in any way.

Pathing metrics, such as Total Time Spent, Entries, and Exits, were originally not available for eVars. However, recent updates have made these metrics available, increasing the value of eVars.

Which to Use

Props: If latency is the largest concern, and you intend to only measure traffic (not success events) with this dimension.

eVars: If data breakdowns and relationships are the largest concerns.



Tip: If you don't want an eVar to persist, you can change its expiration to 'hit' so it doesn't keep any data beyond the hit.

Using Props as Counters

A counter stores (and sometimes displays) the number of times a particular event or process has occurred.

You can use a prop to count the number of times an event occurs. For example, you want to track the use of the Real Player vs. the Windows Media Player on your site. Each page contains **Code to Paste**, in which you can see **s.prop** variables. Use **s.prop 1** to track the players. For page A, enter a value in **s.prop1** to represent Real Player.

```
s.prop1="RealPlayer"
```

For page B, enter a similar value in **s.prop1** for Windows Media Player, as shown below.

```
s.prop1="WindowsMP"
```



Note: Adobe offers up to 75 **s.prop** variables for you to use.

As visitors come to your site and visit the pages containing the Real Player or Windows Media Player, Analytics is able to segment the users based on which pages they visited. The **Custom Traffic** report then shows the number of visits to each page.



Note: The name of the **Custom Traffic** report can be customized. For example, the **Custom Traffic** report can be renamed to "Player Types Report."

Counting Content Hierarchies

A common usage of **content hierarchies** is to show the different paths visitors have taken from a certain page, level, and so forth.

How should I track my content hierarchy?

You must first understand the reporting requirements for tracking **content hierarchies**. If the requirements for tracking the hierarchy are very detailed, often times the hierarchy (*hier*) variable is recommended. Hierarchies usually require a strict, predefined taxonomy where the same child node rarely lives under multiple parent nodes. Consider the following example:

Global Hierarchy

All Sites > Regions > Countries > Language > Category

In this example, the hierarchy could begin to break down at the language level. If a requirement is to report on overall "English" traffic, you can run into the problem where English appears under USA, England, Australia, and so forth. Hierarchies allow you to only drill down. In order to slice horizontally across multiple hierarchies, the best practice is to use a **custom traffic variable** (prop).

If you want to provide users with the ability to drill down through the site (similar to how users would browse the site) and report on **Unique Visitors** at each level of the hierarchy, the hierarchy variable is recommended.

There are occasions when using both props and the *hier* variable makes sense. The following is a supported prop for each variable type:

	Props	Hierarchy
Correlations	✓	✓
Pathing	✓	
Page View	✓	✓
Unique Visitors	✓	✓
Classifications	✓	

What is a Predefined Event?

List of **predefined** events.

prodView	Success event occurs any time a visitor views a product.
scView	Success event occurs any time a shopping cart is viewed.
scOpen	Success event occurs any time a visitor opens a shopping cart for the first time.
scAdd	Success event occurs any time a product is added to a shopping cart.
scRemove	Success event occurs any time an item is taken out of a shopping cart.
scCheckout	Success event occurs on the first page of a checkout.
purchase	Success event occurs on the final page of a checkout (includes Revenue, Orders, and Units).

When one of the predefined events above occurs, an instance of the event is incremented. You can view the metrics related to the event in several different reports. See below for an example of the code used to configure predefined events.

```
s.events="scAdd"
```

```
s.events="scOpen,scAdd"
```

- In the first example above, *scAdd* is the value of the event. Any time an item is added to the shopping cart, the event is incremented.
- In the second example, two values are captured at the same time. When multiple success events occur on the same page, each event is incremented.

Detailed Product View Page

The *products* variable is used for tracking products and product categories (as well as purchase quantity and purchase price).

A success event should always be set in conjunction with the *products* variable.

```
s.events="prodView"
```



Note: While *prodView* is treated in implementation like an event, it does not have the same flexibility in the interface. The *prodView* event is an instance of the product and is only available in the products report. Adobe recommends you use a custom event in addition to the *prodView* event. This way, you can see the product view metrics alongside other metrics in other conversion reports.

```
s.products=";diamond earrings (54321)"
```



Note: The *products* string syntax must begin with a semicolon. This is a legacy syntax requirement. It was previously used to delimit the category and product, but that creates a limitation within the interface should you ever want to change how you are classifying products. For the maximum flexibility in your reporting, it is best to leave this blank and use Classifications to set up categories.

Shopping Cart Page (scOpen, scAdd, scRemove)

```
s.events="scOpen,scAdd"
```

```
s.products=";SKU"
```

First Checkout Page

```
s.events="scCheckout"  
s.products=" ;SKU"
```

Confirmation Page

```
s.events="purchase"  
s.products=" ;SKU"
```



Note: While using the SKU in the product string may make the products report less readable, it provides the maximum flexibility later when you want to classify your products. You can create categories from the SKU that indicate finish, manufacturer, category, and sub-category.

When the *products* variable is set in conjunction with the *purchase* event, the purchase quantity and total purchase price are included in the products value as shown above.

What is a Custom Event?

Custom events let you define the success type that you want to track.

Although similar to **predefined** events, **custom** events let you define your own success metric. For example, if you have a newsletter, your success event could be "Registration." Clearly, "Registration" is not part of the **predefined** events. By using a **custom** event, you can track the number of visitors who register for your newsletter. **Custom** events follow the standard syntax shown below.

```
s.events="event3"
```

The code above shows how to assign an event to the *events* variable. If you do not modify the event name in the interface, then "event3" would display in the interface.

By default, success events are configured as "counter" events. Counter events count the number of times a success event is set. Some success event uses require an event be incremented by some custom amount. These events can be set either as "**numeric**" events or "**currency**" events. You can change the event type using Admin Console.

Hash Collisions

Describes what a hash collision is and how it can manifest.

By default, Adobe treats prop and eVar values as strings, even if the value is a number. Sometimes these strings are hundreds of characters long, other times they are short. To save space, improve performance, and make everything uniformly sized, the strings are not used directly in the processing tables. Instead, a 32-bit or 64-bit hash is computed for each value. All reports run on those hashed values until the data is presented, where each hash is replaced by the original text. Without this compression, reports would likely take minutes to run.

For most fields, the string is first converted to all lower case (reducing the number of unique values). Values are hashed on a monthly basis (the first time they are seen each month). From month to month there is a small possibility that two unique variable values will be hashed to the same hash value. This is known as a *hash collision*.

Hash collisions can manifest in reporting as follows:

- If you are trending a value and you see a spike for a month, it is likely that another values for that variable got hashed to the same value as the value you are looking at.
- The same things happens for segments for a specific value.

Hash Collision Example

The likelihood of hash collisions increases with the number of unique values in a dimension (eVar). For example, one of the values that come in late in the month could get the same hash value as a value earlier in the month. The following example may help explain how this can cause segment results to change. Suppose eVar62 receives "value 100" on February 18. Analytics will maintain a table that may look like this:

eVar62 String Value	Hash
Value 99	111
Value 100	123
Value 101	222

If you build a segment that looks for visits where eVar62="value 500", Analytics determines if "value 500" contains a hash. Because "value 500" does not exist, Analytics returns zero visits. Then, on February 23, eVar62 receives "value 500", and the hash for that is also 123. The table will look like this:

eVar62 String Value	Hash
Value 99	111
Value 100	123
Value 101	222
Value 500	123

When the same segment runs again, it looks for the hash of "value 500", finds 123, and the report returns all visits that contain hash 123. Now, visits that occurred on February 18 will be included in the results.

This situation can create problems when using Analytics. Adobe continues to investigate ways to reduce the likelihood of these hash collisions in the future. Suggestions to avoid this situation are to find ways to spread the unique values between variables, remove unnecessary values with processing rules, or otherwise reduce the number of values per variable.

Frequently Asked Questions about Analytics Implementation

Frequently asked questions about implementation, and links to more information.

Question	Answer
How do I manage Analytics users and groups?	For information about managing users and groups, refer to User Management in the Adobe Analytics help.
eVar Expiration - Why are the eVars getting attributed to 'None' in the reports?	Expire After specifies a time period, or event, after which the eVar value expires (no longer receives credit for success events). If a success event occurs after eVar expiration, the None value receives credit for the event (no eVar was active). If you select an event as an expiration value, the variable expires only if the event occurs. If the event does not occur, the variable never expires. More...
Custom Event Visibility - Why do Custom Events not appear in the reports menu?	In the Visibility column, you can hide standard (built-in) metrics, custom events, and built-in events in the Menu, Metric Selectors, Calculated Metrics Builder, and the Segment Builder. This setting does not impact the data collection for that metric or event; it affects only its visibility in the user interface. More...
Timestamps - What do I need to consider before changing timestamp settings?	Using the Timestamps Optional feature, you can combine non-timestamped data with timestamped data without incurring data loss. Offline data with timestamps generated from a mobile device can be combined with live, non-timestamped data from a web page—or integrated with data from any platform using a client-side timestamp call. More...
Visitor ID - How does the Visitor ID grace period work and how is it enabled?	If you have multiple JavaScript files that are sending data to the same report suite, or if you are using other technologies on your site such as Flash video measurement, we recommend configuring a grace period. More...
Visitor ID - What is the difference between the Experience Cloud Visitor ID and the Analytics Visitor ID?	The Experience Cloud ID service assigns a unique, persistent identifier to all your site visitors. With this ID, visitors and their data can be shared among other solutions in the Experience Cloud. Also, this ID can replace or work with solution-specific IDs such as the Analytics Visitor ID. More...
Visitor ID - How is the Visitor ID set if cookies are blocked?	If the standard s_vi cookie is unavailable, a fallback cookie is created on the domain of the website with a randomly generated unique ID. This cookie, named s_fid, is set with a 2-year expiration and is used as the fallback identification method going forward. More...

<p>Dynamic Tag Management - Why does my DTM rule not fire?</p>	<p>If your event-based rule does not fire, then there is likely an issue with the selector or condition of the rule. Locate the element on your site where the desired event action occurs, right click and select Inspect element. Inspect the highlighted script in the box that opens and ensure you are targeting the correct element. More...</p>
<p>How do I implement Heartbeat Video Tracking?</p>	<p>This section contains instructions on downloading the video heartbeat SDKs and developer guides for your platform. Make sure you also download the developer guide that is in the docs folder when you download the SDK as it contains the specific implementation instructions for video heartbeat.</p>
<p>How do I add cookies to the right subdomain?</p>	<p>The <code>cookieDomainPeriods</code> variable determines the domain on which the Analytics cookies <code>s_cc</code> and <code>s_sq</code> are set by determining the number of periods in the domain of the page URL. This variable is also used by some plug-ins in determining the correct domain to set the plug-in's cookie. More...</p>
<p>Tracking Server - How do I correctly populate my tracking server?</p>	<p>When you configure an implementation to send data to Adobe Analytics servers, you must send it to the correct location. Failure to do so causes inflated visitor counting or data loss. More...</p>
<p>Performance - Can a failure to load the external Adobe JavaScript, whether due to internet connection, proxy, firewall, or service interruption at Adobe, affect performance?</p>	<p>No. The JavaScript file is not hosted on Adobe servers, so an Adobe outage will not affect the JavaScript execution. If dynamic tag management is employed, the JavaScript file is hosted by Akamai, or on a server location determined by customers.</p> <p>See Will Dynamic Tag Management reduce my website's performance? at the Dynamic Tag Management FAQ.</p> <p>Additionally, you can host your own core dynamic tag management file if you are not comfortable relying on Akamai's CDN. See Embed Code and Hosting Options.</p>
<p>Performance - Can the loading of the external Adobe JavaScript cause a reduction in performance?</p>	<p>The JavaScript file is cached in the visitor's browser after it is initially loaded, and is typically downloaded no more than once per session. The file is not downloaded on each page, even though it is used by every page on the site. On most web sites, users average more than a few page views per session, so transferring JavaScript that is used multiple times into this file can result in less overall downloaded data.</p> <p>JavaScript for AppMeasurement Compression: If you are concerned about the page weight (size) of Adobe's JavaScript client, Adobe recommends that you consider compressing the file using GZIP. GZIP is supported by all major browsers and offers better performance than JavaScript compression to compress and decompress the core <code>s_code.js</code> JavaScript file.</p>
<p>Performance - Can the sending of data from the browser to Adobe services reduce performance?</p>	<p>The Adobe JavaScript file creates an image object within the HTML page, and the browser then requests the image object from Adobe servers. If the Adobe servers were to be slow or unresponsive, the thread handling that request would be delayed until the image returns or a timeout occurs. Because browsers handle images with multiple threads, an Adobe outage would have a minimal impact on the page load time, tying up one thread while the others continue to function.</p>

Performance - Can a JavaScript event from Adobe impact system behavior or functionality?	No. See preceding performance answers.
How can I change collected data, based on defined conditions of my own?	Use processing rules to simplify data collection and manage content as it is sent to reporting. learn more
Which is the latest version of s_code file?	This section contains a release history for AppMeasurement libraries across web and mobile platforms. The latest version of each library can be downloaded in Reports & Analytics > Admin Tools > Code Manager. learn more
How do I debug s_code file?	The Adobe Debugger (previously DigitalPulse Debugger) is a free tool provided by Adobe that lets you view the data being collected from your site on any given page. learn more
How do I track different link types?	File downloads and exit links can be automatically tracked based on parameters set in the AppMeasurement for JavaScript file. learn more
How do I track video ?	JavaScript can be used to track a wide variety of players. To track using JavaScript, you add code to the web page that contains your player and track the player using event handlers. learn more
How do I track a mobile App?	Acquisition links with unique tracking codes can be generated in Adobe Mobile services. When a user downloads and runs an app from the Apple App Store after clicking on the generated link, the SDK automatically collects and sends the acquisition data to Adobe Mobile services. iOSAndroid
How do I implement video tracking?	You can track media players by creating functions attached to the video player event handlers This lets you call Media.open, Media.play, Media.stop, and Media.close at the appropriate times. learn more
How do I set up the First Party Cookie?	Analytics uses cookies to provide information on variables and components that do not persist between image requests and browser sessions. These harmless cookies originate from a domain hosted by Adobe, known as third-party cookies. learn more
How do I get an SSL certificate?	Determine whether your site uses https:// protocol. If it does, requesting a CSR and purchasing an SSL certificate is required. Note: An SSL certificate is not required if you do not have any secure pages or content. This entire step may be skipped if you use only http:// protocol on your site. learn more
Where do I find information about the certification expiration notice?	SSL certificates expire each year, meaning that Adobe requires an updated certificate request each time this happens. The FPC specialist provides sufficient warning when this occurs, however, it is recommended to be proactive in monitoring the expiration and providing Adobe with this updated certificate. learn more
What are plugins?	AppMeasurement for JavaScript plug-ins are programs or functions that perform several advanced functions. These plug-ins extend the capabilities of your JavaScript file to give you more functionality that is not available with a basic implementation. Adobe offers a number of other plug-ins as part of advanced solutions. Contact your Account Manager if you want to capture data using JavaScript but are unsure how to proceed. learn more
Information about data insertion API?	Adobe has created multiple ways to send data into Analytics. learn more

What is a 500 error?

Information about the internal server error which caused a "500 Query Error" status.
[*pageType variable*](#)

Prepare to Implement Analytics

There are tasks you should complete and information you should gather when preparing to implement Analytics.

The information and documents in this section are gathered directly from Adobe Consulting. Following these guidelines and filling out the provided questionnaires will help you implement Analytics, whether you implement by yourself or work with Adobe Consulting (recommended).

If you are working with an Adobe Consultant, it is not required that you know the information in these documents before implementing Analytics. However, being familiar with the documents Consulting will create and the questions they will ask can help to speed the pre-implementation process.

Business Requirements Document

The Business Requirements Document (BRD) is used to document the key business objectives and requirements.

Consulting will work with you to compile the BRD. The contents of this document depend on a number of factors, such as the nature of your site or application, and your business vertical (such as retail, media, travel, or lead gen).

Consulting will also work with you to compile a Solution Design Reference (SDR), which documents the solution architecture, and a validation checklist to validate the implementation.

Business Governance Questionnaire

There are six pillars of digital analytics that can help you understand whether your organization is positioned to make the most of the business intelligence your data can provide.

- Clear objectives
- Organization-wide scope
- Right expertise
- Tools and technology
- Process
- Governance (that ties them all together)

Organizations that successfully make the most of their data have put into place well-defined and well-communicated roles and responsibilities. This holds teams and people accountable across the full spectrum of activities required to collect, analyze, and use data to measure and act on business goals. This is known as governance.

Responsibility is another aspect of governance. Governance is the process of outlining who has responsibility for making collection, analysis, implementation and assessment a success and how these activities can be supported. This can range from ownership and responsibility of a particular area to simply being informed and aware of the progress or setbacks. The [Business Governance Questionnaire](#) contains business questions that need to be filled by our client before the Analytics implementation kick-off meeting.

Technical Questionnaire

Adobe uses proven methodologies and enlists professional help to guide our clients to accurately plan in the initial stages of the project and to successfully complete the implementation.

The [Technical Pre-Implementation Questionnaire](#) contains technical questions, which should be filled out before the Analytics implementation kick-off meeting.

Technical Specification

Your Adobe Consultant will discuss the technical specifications of your implementation with you.

It can be useful for you to be familiar with the content of the [Technical Specifications](#) document used by Consulting, although it is not required for you to know this information if you are working with a Consultant. This document is used as the standard guide for deploying Adobe Analytics on a given site. The document is divided into different sections, each of which describes a different component of the overall Analytics solution.

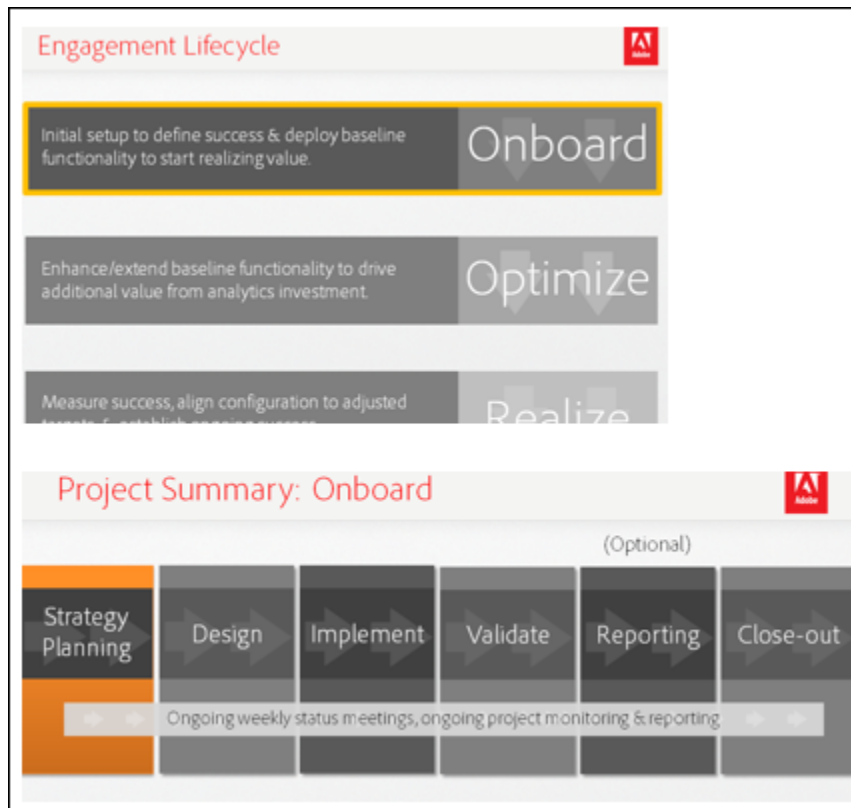
The industry specific Tech Specs cover the fundamental solutions and can be leveraged as a starting point for creating a more complete document.

Review Website or Application

A site or application walk-through is a key step in the implementation process, in which you lead your Implementation Consultant through the site and explain the technical factors and business flows.

This step should be carried out whenever there is a requirement for implementation or implementation changes.

- This step happens in the "strategy planning" phase, in the "onboard" process of an engagement lifecycle.
- This step typically takes place after the kick-off call between your organization and Consulting,



Prerequisites

It is important to work with your implementation consultant to provide a good understanding of your business and the overall objective of your site or application. Some of the places best suited to give some background information are:

- Contract/Pre-Kick-off call with Sales Team
- Company website (For example, the About and News pages)
- Review Industry vertical playbook & related resources
- Review important concepts in Wikipedia (such as Revenue, Affiliates, and Competitors)
- review Play Store and App Store descriptions and reviews

Key Stakeholders

Adobe:

- Implementation Consultant
- Business Consultant
- Solution Architect
- Project Manager

Client:

- Site Master
- Business Lead
- Technical Lead

Methodology

- The site/app walkthrough happens over a phone with screen share or on a site visit
- Your Consultant will ask you to demonstrate the key business flows that impact revenue
- The Consultant will observe, record, and document the outcomes

The [Adobe Pre-Implementation questionnaire](#) and [Digital Assets Governance](#) document contain the questions that you should answer during the walkthrough session.

Key Outcomes

The following information should be known following the website or app review:

Business:

- Purpose and the reason why the site exists
- Strategic importance of the site
- Relationships or groups with other internal/external sites (if any)
- Site sub-functions (service or account details, or referral)
- Key business processes that a visitor will go through

Technical:

- Internal and external domain names
- Secure content (if any)
- Hosted conversion process

- Currency and time zone settings
- Hybrid app details

Dashboard Creation

As part of the implementation process, your Adobe Consultant will help you create a dashboard.

Your Consultant will schedule and hold a dashboard kickoff meeting. During the meeting, You and your Consultant will gather requirements and fill out the Dashboard Requirements Document (DRD). After the call, your Consultant will give you a final copy of the DRD and identify the next steps.

Adobe Analytics Dashboard Fundamentals

Dashboards help your business:

- Monitor key performance indicators (KPIs)
- Understand your customers better
- Respond to competitive pressures more quickly
- Assist in quick decision making and planning

Before Adobe can help your organization with the visual design of a dashboard, it is important to help your Consultant understand your requirements. The following questions should be answered:

1. How often should the data be updated in the dashboard?
2. Who will use the dashboard? Is it for a single person, a single group, or people in several different departments?
3. What will they use the dashboard to do? What questions will they use it to answer? What actions will they take in response to these answers?
4. What specific information should be displayed on the dashboard? List all of the data items that should be included on the dashboard. Also indicate the level of summary/detail at which each item should be expressed on the dashboard.
5. Which of the data items listed above are the key (most important) items needed to meet the objectives supported by the dashboard?
6. What are the logical groupings that could be used to organize these data items on the dashboard? Into which of these groups does each data item belong?
7. What are the useful comparisons that will allow the dashboard's users to see the data items listed above in context?

The [Adobe Analytics Dashboard Fundamentals](#) document contains a list of questions that will help Adobe create an initial dashboard structure. Once in place, you and your Consultant can customize the dashboards according to your business needs.

Choose an implementation method

There are multiple ways to implement Adobe Analytics.

- Launch (Recommended)
- Dynamic Tag Management
- JavaScript

Launch

Launch is the next generation of website tag and mobile SDK management capabilities from Adobe. Launch gives you a simple way to deploy and manage all of the analytics, marketing, and advertising integrations necessary to power relevant customer experiences.

Launch empowers anyone to build and maintain their own integrations with Launch, called Extensions. These extensions are available to web and mobile Launch customers in an app-store experience, so customers can quickly install, configure, and deploy their integrations.

For more information, see [Getting Started with Launch](#).

Dynamic Tag Management

Dynamic Tag Management (DTM) automates much of the detail work required to implement Analytics. Enter the required information in a form-based interface, and Dynamic Tag Management generates the code you need to add to your pages.

Although Dynamic Tag Management greatly simplifies the implementation process, you should be familiar with [basic](#) before you begin. It is also useful to be familiar with JavaScript.

For example, you should understand the following:

- What an [eVar](#) is and how it works
- When to use a [custom event](#)

For information about getting access to Dynamic Tag Management and getting up and running, see [Getting Started](#) in the Dynamic Tag Management Product Documentation.

For more information, see [Implement Analytics with Dynamic Tag Management](#).

JavaScript

The JavaScript implementation method requires you to configure the JavaScript codes on your pages manually. Much of this effort can be simplified if you use the Dynamic Tag Management implementation method. However, some users might require the JavaScript method.

The JavaScript implementation requires:

- Strong JavaScript skills
- Solid understanding of Analytics concepts and terminology

For more information, see [Implement Analytics using JavaScript](#).

Getting Started with Analytics Implementation


Learn about the first-time customer experience for implementing Adobe Analytics implementation.

New users can quickly create your first Analytics report suite (data repository) using this *Getting Started with Adobe Analytics* setup modal. Then, you can deploy Analytics code using Dynamic Tag Management.

Dynamic Tag Management allows you to manage your Adobe Analytics implementation without needing to make changes to your site every time. If you're implementing a Mobile app, you can get the SDK that you need to begin collecting valuable data from your apps.

In this procedure enables you to:

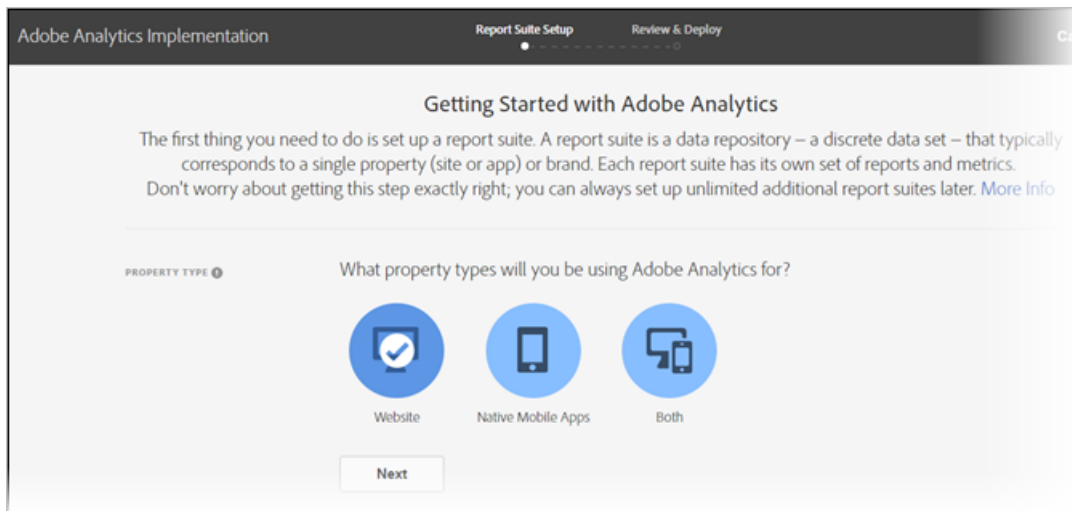
- Quickly create your first [report suite](#).
- Deploy Analytics and the [Experience Cloud ID Service](#).
- Run reports on basic page-level data.

 **Note:** Before you begin, verify that Analytics is [enabled in the Adobe Experience Cloud](#) (the solution provisioning process). If you received an email inviting you to log in to Analytics in the Enterprise Dashboard, you've completed this prerequisite.

To run the simplified implementation modal

1. Log in to the Adobe Experience Cloud (<your company>.marketing.adobe.com).

When you access Analytics, the system determines if you have a report suite. If not, the **Getting Started with Adobe Analytics** page displays.



Alternatively, you can run this setup in Analytics by clicking **Help > Welcome to Adobe Analytics**.

2. Specify the following basic information about your business:



Element	Description
Property Type	Is your implementation for web, mobile, or both?
Industries	Specify how your company makes money (products, customer services, leads, brand awareness, and ads).

Element	Description
Data Layer	(Recommended) A JavaScript array which is used to store information. If you perform the automatic setup using Dynamic Tag Management, you will be using a data layer. For a blog on data layers, see Data Layer: From Buzzword to Best Practice .
Data Repository (Report Suite)	A report suite is a discrete data set that typically corresponds to a single property (site or app) or brand. Each report suite has its own set of reports and metrics.
Time Zone	Your local time zone. (Automatically detected.)
Estimated Page Views	Roughly the number of page views your site receives per day.
Base Currency	The currency in which you do business.

3. Click **Next**.

The system creates a report suite.

4. To begin deployment, click **Next**, then click one of the following options:

Element	Description
Deploy	Launches Dynamic Tag Management, where you can log in and deploy Analytics. This process automatically implements the <code>AppMeasurement.js</code> file and the Experience Cloud ID service (<code>VisitorAPI.js</code>).  Important: In a new browser tab, a help page is displayed that walks you through Adobe Analytics deployment via Dynamic Tag Management.
Download	Downloads the installation file, named <code>INSTALL-ME <report suite name>.js</code> . This option is for experienced users who understand JavaScript Implementation .  Important: Downloading the code does not constitute deploying Analytics. This is a manual deployment that you perform on the pages of your site, or through Adobe consulting services.

5. Run a report.

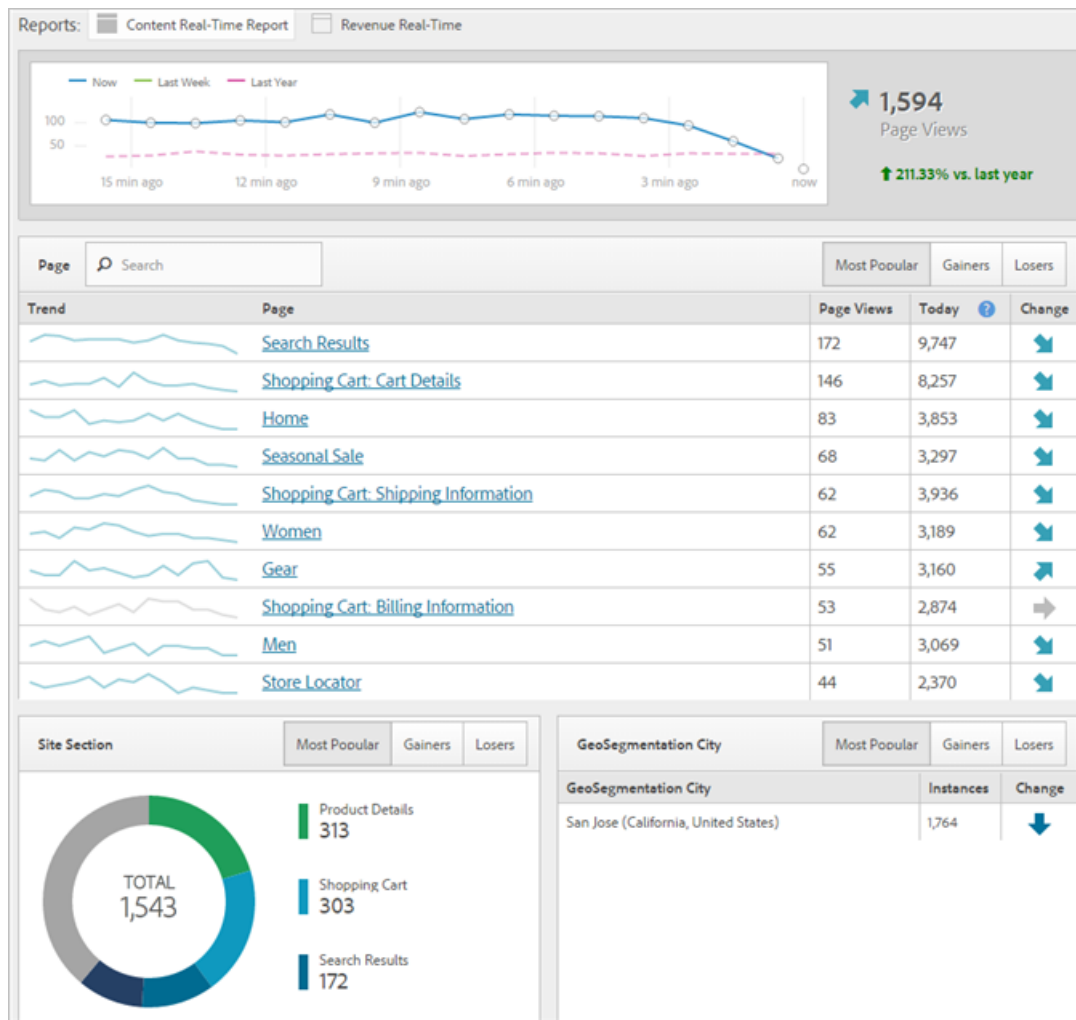
After deploying the Analytics tool, you can run a report in Reports & Analytics to confirm that data is coming to your site. (See [Sign in and Navigate](#) to get familiar with the Analytics interface.)

For example, a **Site Metrics > Real-Time** lets you see immediate data.



Note: The Real-Time report requires some configuration prior to running. See [Configure Real-Time Report](#).

Example Real-Time Report



Implement Analytics with Dynamic Tag Management

You can use Dynamic Tag Management (DTM) to implement Adobe Analytics.

Use Dynamic Tag Management to manage tags and collect and distribute data across digital marketing systems. Dynamic Tag Management provides a single data layer that pushes data from multiple sources. Dynamic Tag Management also enables responsive delivery of user-specific content.

This help section provides specific information about using Dynamic Tag Management to implement Adobe Analytics. For detailed information about Dynamic Tag Management, refer to the [Dynamic Tag Management Product Documentation](#). For information about accessing DTM and common tasks when starting to use DTM, see [Getting Started](#) in the Dynamic Tag Management Product Documentation.

For a detailed summary of the Dynamic Tag Management implementation steps, refer to [Deploy Adobe Analytics Using Dynamic Tag Management](#) in Getting Started with Adobe Analytics.

Overview of Implementation Steps

This guide leads you through the following steps to implement Analytics using DTM:

1. [Create a web property.](#)
2. [Configure your hosting options.](#)
3. [Add header and footer code to each managed page.](#)
4. [Add the Adobe Analytics Tool.](#)
5. Create [data elements](#), [rules and conditions](#), and [actions](#).
6. Publish tools and rules to the production server.

Create Web Property

A web property can be any grouping of one or more domains and subdomains with a library of rules, included in one embed code.



Note: Only a user with Admin rights can create a property. For more information about roles, see [Create and Manage Groups in DTM](#) in the Dynamic Tag Management Product Documentation.

You can manage and track these assets with DTM. For example, suppose that you have multiple websites based on one template and you want to track the same assets on all of these websites. You can apply one web property to multiple domains.



Tip: Plan ahead to determine how many web properties you need. See [Getting Started with Adobe Dynamic Tag Management](#).

For general information about web properties and best practices, see [Web Properties](#) in the Dynamic Tag Management Product Documentation.

1. Navigate to your company page, then click **Add Property**.

Create Property

Name *

URL *

This site spans multiple domains

- Advanced Settings

Allow Multi-Rule Approvals

Enable Selective Publish

Tracking Cookie Name ?

Timing Defaults

Tag Timeout ? **Anchor Delay** ?

Milliseconds **Milliseconds**

2. Fill in the fields:

Element	Description
Name	The name of your property.
URL	The base URL of the property.
This site spans multiple domains	<p>You can add and remove domains if you want visitor data to persist between domains. If this option is selected, data for the visit persists across subdomains.</p> <p>This setting lets you specify how you would like to track traffic moving between your associated subdomains or domains. Links to subdomains are treated as outbound links. Visits to subdomains are tracked separately.</p>

3. (Optional) Configure **Advanced Settings**.

Element	Description
Allow Multi-Rule Approvals	Allows multiple rules for this property to be approved at one time. The default approval allows only single-rule approval.
Enable Selective Publish	Specifies whether to allow users to select which approved rules are published. This is the default option.

Element	Description
Tracking Cookie Name	Overrides the default tracking cookie name. You can customize the name that Dynamic Tag Management uses to track your opt-out status for receiving other cookies.
Tag Timeout	<p>Specifies how long Dynamic Tag Management waits for a tag to fire before timing out and cancelling the tag request.</p> <p>Because of how Dynamic Tag Management works, don't worry about this being a high number. DTM has effective methods of ensuring that slow tags do not affect the user experience.</p>
Anchor Delay	<p>Specifies how long Dynamic Tag Management waits for tags to fire on clicked links before moving to the next page. The default value is 100 milliseconds.</p> <p>Longer delays improve tracking accuracy. Adobe recommends a delay of 500 milliseconds or less, which the user will not perceive.</p> <p>Dynamic Tag Management will wait up to the time specified, but if the beacon fires sooner, the delay is cut short. (That is, user won't always wait the full length of the delay.)</p>

4. Click **Create Property**.

Configure Hosting Options

You can deploy Dynamic Tag Management using one or more of the available hosting options.

Dynamic Tag Management provides a number of options to host the required JavaScript files.

For detailed information about hosting, see [Embed Code and Hosting Options](#) in the Dynamic Tag Management Product Documentation.

On the Embed tab, select a hosting option.

Hosting Option	Description	Implementation
Akamai	<p>The simplest hosting option to implement.</p> <p>Globally distributed delivery network.</p> <p>Adds additional third-party infrastructure dependencies (DNS lookup, Akamai availability).</p> <p>For more detailed information, see Akamai in the Dynamic Tag Management Product Documentation.</p>	<ol style="list-style-type: none"> 1. Dynamic Tag Management generates custom JavaScript libraries. 2. Dynamic Tag Management exports the custom JavaScript libraries to Akamai. 3. The target website references the Akamai-hosted Dynamic Tag Management libraries directly at the page level.

Hosting Option	Description	Implementation
Self-hosting: FTP Delivery	<p>A <i>push</i> approach, whereby Dynamic Tag Management exports custom JavaScript libraries directly to the web content server host via the FTP protocol.</p> <p>This solution requires an FTP server and credentials to be available on the web content server to publish changes to the custom Dynamic Tag Management libraries.</p> <p>For more detailed information, see FTP in the Dynamic Tag Management Product Documentation.</p>	<ol style="list-style-type: none"> Dynamic Tag Management generates custom JavaScript libraries. Dynamic Tag Management exports the custom JavaScript libraries to host server via FTP. The target website locally references the custom Dynamic Tag Management libraries.
Self-hosting: Library Download	<p>A <i>pull</i> approach, whereby the application exports custom JavaScript libraries. There, the libraries can be accessed by a hosted server-side process.</p> <p>Additionally, the libraries are available via web download directly from the Dynamic Tag Management interface.</p> <p>This solution requires either a manual retrieval and publication of the Dynamic Tag Management libraries or the creation of an automated process that pulls the libraries from Akamai onto the web content server.</p> <p>This approach takes the most time to set up, but is also the most secure and flexible option.</p> <p>To check if the latest version of your library file is being referenced, use the command</p> <p>For more detailed information, see Library Download in the Dynamic Tag Management Product Documentation.</p>	<ol style="list-style-type: none"> Dynamic Tag Management generates custom JavaScript libraries. Dynamic Tag Management exports the custom JavaScript libraries to Akamai. Custom Dynamic Tag Management libraries are manually or programmatically moved to the web content server. The target website locally references the custom Dynamic Tag Management libraries.

You can use more than one hosting option. For example, you might host your staged files using Akamai, but self-host your production site. Note that each hosting type has its own embed code, and only one embed code can be added to a page.

Add Header and Footer Code

Use Dynamic Tag Management to add header and footer code that determines the loading of JavaScript and page content on your site. You must install both the header and footer code on every page of your site, regardless of the hosting option used.

Because Dynamic Tag Management includes a snippet of code in both your header and footer, you can run rules at the beginning or end of a page. This ability allows you to implement testing tools and other technologies while retaining control over tracking your pages.

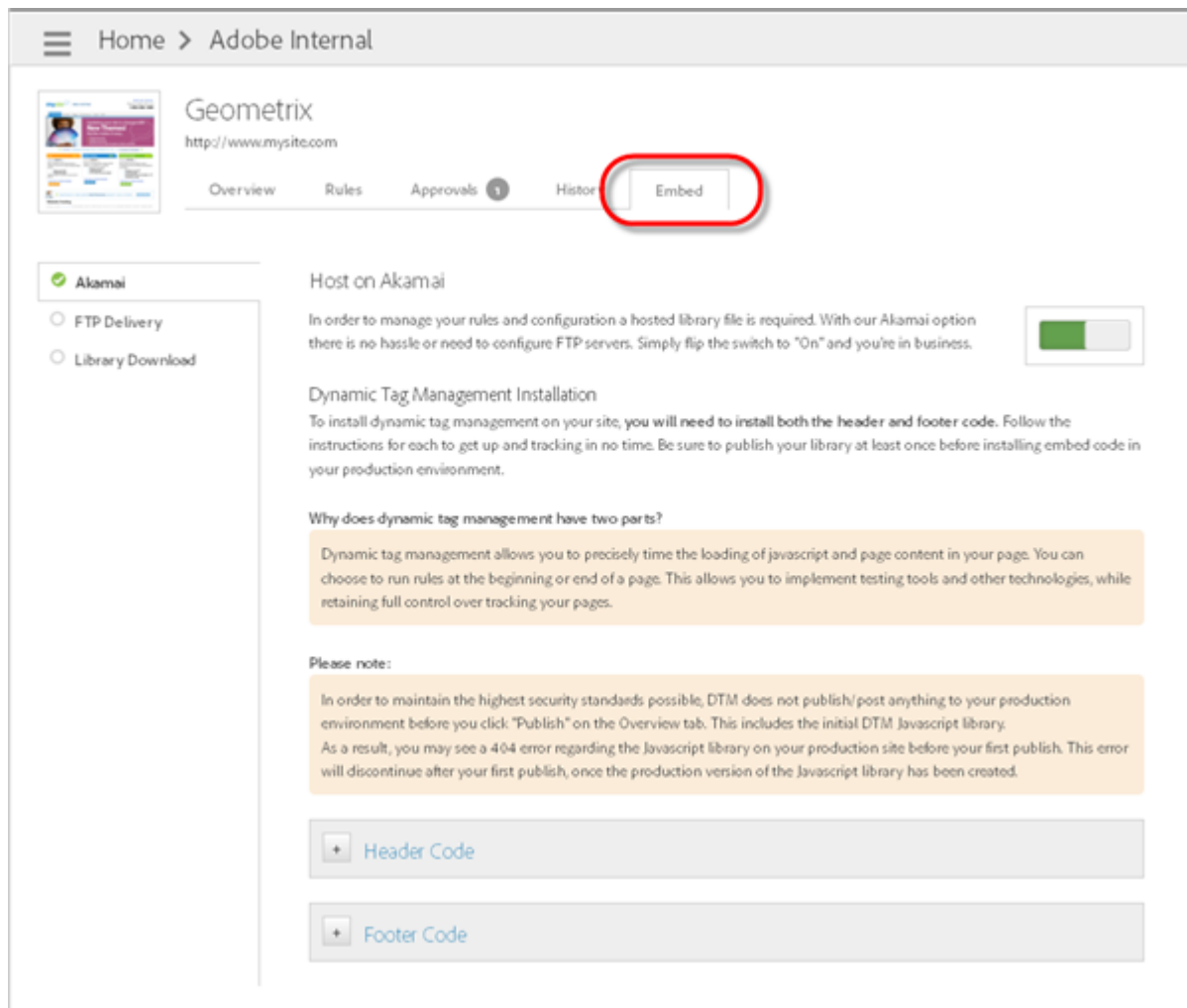
Dynamic Tag Management creates staging and production embed codes you can use to test your changes in your staging environment before pushing changes to your production environment.



Important: *For a successful implementation, it is critical that you follow these instructions as they appear in Adobe Help. Specifically, you must place the header code in the `<head>` section of your document templates. Also, you must place the footer code just before the closing `</body>` tag. Placing either of these embed codes elsewhere in your markup, or using asynchronous methods to append the embed codes, or wrapping the embed codes in any way, are not a supported implementations of Dynamic Tag Management. The embed codes must be implemented exactly as provided.*

An unsupported implementation will yield unexpected results and prevent Customer Care and Engineering from assisting with your implementation.

1. In the Dynamic Tag Management interface, open the **Embed** tab and select your hosting option (such as Akamai), then toggle the switch to "On."
2. Copy the production header code provided in the Embed tab of Dynamic Tag Management and place it within the `HEAD` section of your site HTML.



Home > Adobe Internal

Geometrix
http://www.mysite.com

Overview Rules Approvals 1 History **Embed**

Akamai
 FTP Delivery
 Library Download

Host on Akamai

In order to manage your rules and configuration a hosted library file is required. With our Akamai option there is no hassle or need to configure FTP servers. Simply flip the switch to "On" and you're in business.

Dynamic Tag Management Installation

To install dynamic tag management on your site, you will need to install both the header and footer code. Follow the instructions for each to get up and tracking in no time. Be sure to publish your library at least once before installing embed code in your production environment.

Why does dynamic tag management have two parts?

Dynamic tag management allows you to precisely time the loading of javascript and page content in your page. You can choose to run rules at the beginning or end of a page. This allows you to implement testing tools and other technologies, while retaining full control over tracking your pages.

Please note:

In order to maintain the highest security standards possible, DTM does not publish/post anything to your production environment before you click "Publish" on the Overview tab. This includes the initial DTM Javascript library. As a result, you may see a 404 error regarding the Javascript library on your production site before your first publish. This error will discontinue after your first publish, once the production version of the Javascript library has been created.

Header Code

Footer Code

Place the code as close to the <head> tag as possible. This code snippet should be placed on every page of your live production site.

Note: Production embed code reflects only the published items in that [property](#). However, embed code for staging reflects all items in the associated property, regardless of the published or unpublished state. To test unpublished items on your production site, locally enable staging in the console by following the instructions in [Test Unpublished Rules for Akamai Hosting](#).

3. Copy the production footer code and place it in the BODY section of your site HTML.

Place the code as close to the </body> tag as possible.

4. Copy the staging header and footer code, then repeat the steps above on your staging site.

Note: The difference between production and staging code snippets is the addition of `-staging` to the filename in the staging version. The footer code remains the same in staging and production.

Verify Header and Footer Code

Verify that your Dynamic Tag Management library is loading properly on your site.

1. Open your site in your browser.
2. Open the **Developer Console** by right-clicking and choosing **Inspect Element > Console**.
3. Press **Enter**.

If the code was properly installed, you will see `true` display in the console.

If the code was not properly installed, you will see the reference error:

```
_satellite is not defined
```

If you receive this error, ensure that:

- You have included the full header code on every page of the site in the `HEAD` section, as close to the `<head>` tag as possible.
- You do not have unexpected characters appearing in the code snippet, potentially as a result of copying and pasting from a formatted document.

Add Adobe Analytics Tool

Deploy Adobe Analytics (Standard and Premium) using Dynamic Tag Management by creating the Adobe Analytics tool and configuring the page code either automatically or manually. The automatic method is recommended for most users.



Note: For improved visitor tracking, it is strongly recommended that you enable [Experience Cloud ID Service](#).

This section contains the following information:

- [Add an Adobe Analytics Tool](#)
- [Edit an Existing Adobe Analytics Tool](#)

Add an Adobe Analytics Tool

1. Click **Web Property Name > Overview > Add a Tool > Adobe Analytics**.

Add a Tool

Tool Type

Tool Name

Configuration method
 ?

Authenticate via:
 Marketing Cloud Web Services ?

Web Services Username

Shared Secret

2. Fill in the fields:

Element	Description
Tool Type	The type of tool, such as Adobe Analytics.
Tool Name	A descriptive name for this tool. This name displays on the Overview tab under Installed Tools .
Configuration Method	<p>Automatic (Recommended): Use dynamic tag management to manage the configuration. This method enables automatic synchronization of Adobe Analytics report suites via a Experience Cloud login or Web Services ID, and manages the AppMeasurement code.</p> <p>After the accounts are connected, Dynamic Tag Management pulls the Adobe Analytics report suite IDs and names into the tool configuration interface, allowing for increased speed in tool deployment with less possibility for user errors.</p> <p>Fill in the fields specific to automatic configuration:</p> <ul style="list-style-type: none"> • Experience Cloud: (Default) Uses Experience Cloud single sign-on. Specify your Experience Cloud ID and password. • Web Services: Specify your Web Services username and shared secret. <p>Shared secret credentials are located in Admin Tools > Company Settings > Web Services.</p> <p>Developers, see Get Web Service Access to the Enterprise API for help with obtaining Web Services credentials.</p>

Element	Description
	<p>Manual: Manually manage the AppMeasurement code. You can download the AnalyticsAppMeasurement code from Admin Tools > Code Manager.</p> <p>Click JavaScript (new) for information about downloading the code locally to copy and paste in the Edit Code field in Library Management.</p> <p>Fill in the fields specific to a manual configuration:</p> <ul style="list-style-type: none"> • Production Account ID: (Required) Your production account for data collection. For Analytics, this is your report suite ID. Dynamic Tag Management automatically installs the correct account in the production and staging environment. • Staging Account ID: (Required) Used in your development or test environment. For Analytics, this is your report suite ID. A staging account keeps your testing data separate from production. • Tracking Server: Specify the information for your tracking server. The Tracking Server and SSL Tracking Server variables are used for first-party cookie implementation to specify the domain at which the image request and cookie is written. For more information, see the Correctly Populate the trackingServer and trackingServerSecure Variable article. • SSL Tracking Server: Specify the information for your SSL tracking server.

3. Click **Create Tool** to create the tool and display it for editing.


Tools are displayed on the **Overview** tab, under **Installed Tools**.

4. (Conditional) Configure the tool further as necessary by following the directions in the links below (**General**, **Library Management**, **Global Variables**, **Pageviews & Content**, **Link Tracking**, **Referrers & Campaigns**, **Cookies**, and **Customize Page Code**).


See [Frequently Asked Questions About the Adobe Analytics Tool](#) for additional information about this tool.

Edit an Existing Adobe Analytics Tool

You can edit an existing Adobe Analytics tool to change its configuration settings.

1. Click the  icon next to an installed tool from the **Overview** tab.
2. Edit the fields as desired.

The following table includes only those elements that differ from the elements available when you are creating an Analytics tool, as described above. However, you can change any element on the page, as described in both tables.

Element	Description
Enable Automatic Configuration	 Note: Enabling this setting changes a manually configured implementation to the automatic configuration method described in Configuration Method .

Element	Description
	<p>This option lets Dynamic Tag Management automatically retrieve your Adobe Analytics account's configuration.</p> <p>The latest available AppMeasurement code is used and upgrade notifications are displayed for selection as new versions become available. You can also roll back to previous AppMeasurement versions as necessary, such as for compatibility reasons. Up to five previous versions are displayed.</p>
Update Credentials	Refresh the API, for example, to update report suites associated with a user.

- (Conditional) Configure the tool further as necessary by following the directions in the links below (**General**, **Library Management**, **Global Variables**, **Pageviews & Content**, **Link Tracking**, **Referrers & Campaigns**, **Cookies**, and **Customize Page Code**).
- Click **Save Changes**.

General

Field descriptions for the General settings in dynamic tag manager, for deploying Adobe Analytics.


Property >  **Edit Tool** > **General**

Element	Description
Enable EU compliance for Adobe Analytics	<p>Enables or disables tracking based on the EU privacy cookie.</p> <p>When a page is loaded, the system checks to see if a cookie called <code>sat_track</code> is set (or the custom cookie name specified on the Edit Property page). Consider the following information:</p> <ul style="list-style-type: none"> If the cookie does not exist or if the cookie exists and is set to anything but <i>true</i>, the loading of the tool is skipped when this setting is enabled. Meaning, any portion of a rule that uses the tool will not apply. <p>If a rule has analytics with EU compliance on and third-party code, and the cookie is set to <i>false</i>, the third-party code still runs. However, the analytics variables will not be set.</p> <ul style="list-style-type: none"> If the cookie exists but it is set to <i>true</i>, the tool loads normally. <p>You are responsible for setting the <code>sat_track</code> (or custom named) cookie to <i>false</i> if a visitor opts out. You can accomplish this using custom code:</p> <pre><code>_satellite.setCookie("sat_track", "false");</code></pre> <p>You must also have a mechanism to set that cookie to <i>true</i> if you want a visitor to be able to opt in later:</p> <pre><code>_satellite.setCookie("sat_track", "true");</code></pre>

Element	Description
Character Set	Displays the available character encoding sets.
Currency Code	Displays the supported currency codes for selection.
Tracking Server	The domain at which the image request and cookie is written. See trackingServer .
SSL Tracking Server	The domain at which the image request and cookie is written. Used for secure pages. If not defined, SSL data goes to trackingServer . See trackingServerSecure .
Data Center	The Adobe data center used for data collection.

Library Management

Descriptions of the fields and options in the Library Management settings in Dynamic Tag Management.

Property >  **Edit Tool** > **Library Management**



Note: If more than one Adobe Analytics tool is used in a single web property, each tool must have a unique tracker variable name. Duplicative object variable names between Adobe Analytics tools within a single web property will cause conflicts.

Element	Description
Page code is already present	Prevents Dynamic Tag Management from installing Adobe Analytics page code if the code is already present on your site. This feature allows you to use Dynamic Tag Management to add to your existing implementation rather than starting from scratch. Be sure to properly set your tracker variable name when checking this box.
Load library at <Page Top or Page Bottom>	Specifies where and when to load the page code. Regardless of your selection, any rules using the Analytics tool will need to have the same setting.
Managed by Adobe (Recommended)	Enable Dynamic Tag Management to manage your library. If you select this option, the following option becomes available: Library Version: Select the latest version from the Library Version menu. Dynamic tag management notifies you when new versions are available. You can revert to a previous version as necessary.
Custom	You can configure the library code.

Element	Description
	<p>If you select this option, the following options become available:</p> <p>Set report suites using custom code below: When this box is checked, Dynamic Tag Management looks for a variable in your custom code called <code>s_account</code>. This variable should contain a comma-separated list of the report suites to which you want to send data.</p> <p>Code Hosted: Choose an option to host the <code>s_code</code>:</p> <ul style="list-style-type: none"> • In DTM: You can host the <code>s_code</code> within Dynamic Tag Management. Click Edit Code to cut and paste the file directly into the editor. • URL: If you have a good <code>s_code</code> file and are happy with the process of updating it, you can provide the URL to the file here. Dynamic tag management then consumes that <code>s_code</code> file for its implementation of Adobe Analytics. <p>Open Editor: Lets you <i>insert core AppMeasurement code</i>. This code is populated automatically when using the automatic configuration method described in Add Adobe Analytics Tool.</p> <p>Tracker Variable Name: If you want to run two instances of Adobe Analytics in parallel (one within Dynamic Tag Management and one natively), you can rename the main <code>s</code> object. Renaming the object name avoids collisions.</p>

Insert Core AppMeasurement code

Insert AppMeasurement code when manually deploying Dynamic Tag Management in Adobe Analytics.

1. On the Adobe Analytics tool page, expand the **General** section, then click **Open Editor**.
2. Unzip the `AppMeasurement_JavaScript*.zip` file you downloaded in [deploy Adobe Analytics](#).

If you opt for custom library, when you open the window it will already have the most recent code version present. There is no need to download the zip from the Admin Console.

3. Open `AppMeasurement.js` in a text editor.
4. Copy and paste the contents into the **Edit Code** window.

```

1  var s = new AppMeasurement();
2  s.account = "[insert account name]";
3
4
5  /*
6  ----- DO NOT ALTER ANYTHING BELOW THIS LINE ! -----
7
8  AppMeasurement for JavaScript version: 1.1
9  Copyright 1996-2013 Adobe, Inc. All Rights Reserved
10 More info available at http://www.omniture.com
11 */
12 function AppMeasurement() { var s=this; s.version="1.1"; var w=window; if(!w.s_c_in)w.s_c_il=[],w.s_c_in=0;
13 } return s; return s.split("").join(""); s.escape=function(b) { var a,c; if(!b) return b; b=encodeURIComponent(
14 !/^[\0-9.\-]+$/i.test(b))&&(a=parseInt(a):2,a>2?a:2,c=b.lastIndexOf("."),c>0){for(;c>0&&c>1;)c=b.last
15 ""?parseInt(d?d:0):-60)?(c=new Date,c.setTime(c.getTime()+f*1E3)):c=1&&(c=new Date,f=c.getYear(),c.set
16 if(f&&f=="prerender"){if(!s.0){s.0=1;for(c=0;c<g.length;c++){s.d.addEventListener(g[c],function() {var b
17 });s.setAccount=s.sa=function(b) {var a,c; if(!s.B("setAccount",arguments))if(s.account=b,s.allAccounts
18 i[j].length==i[j]&&(i==10);if(!i&&(f=="&&(f+="&"+b+"."),j=a[a],a&&(g=g.substring(d.length),g.length>
19 q;w="list"?g="l"+g;w="hier"&&(g="h"+g,j=j.substring(0,255))}}f+="&"+s.escape(g)+"="+s.escape(j)}}f!=""
20 (m+(m!="?"",":")+s.events2)}for(c=0;c<a.length;c++){d=a[c];f=s[a];e=d.substring(0,4);g=d.substring(4
21 255)s.pageURLRest=f.substring(255),f=f.substring(0,255);break;case "pageURLRest":d="-g";break;case "ref
22 "vvp";break;case "currencyCode":d="cc";break;case "channel":d="ch";break;case "transactionID":d="xact":
23 f.split(",");f="";for(e=0;c<g.length;e++){j=g[e],w=i.indexOf("="),w>=0&&(j=j.substring(0,w)),w=i.indexOf

```

5. Adobe recommends adding the following code above the *Do Not Alter Anything Below This Line*:

```

var s_account="INSERT-RSID-HERE"
var s=s_gi(s_account)

```



Important: If you add this code, it is recommended that you also select the **Set report suites using custom code below** checkbox in the overall library settings.

6. Click **Save and Close**.

If you are using the Media Module, Integrate Module, or implementation plug-ins, you can copy them into the code section as well. The managed code in Dynamic Tag Management can be configured exactly like the JavaScript file in a typical implementation.

Global Variables

Field descriptions and information about variables when using Dynamic Tag Management to deploy Adobe Analytics.

These variables fire on all page load rule beacons. You can accomplish the same effect by using a [Page-Load rule](#) set to fire on all pages. These variables might not fire in [Direct-Call](#) and [Event-Based](#) rules.

Global Variables - Field Descriptions


Property > **Edit Tool** > **Global Variables**

Element	Description
Server	The predefined variable populates the Servers report in Adobe Analytics. See server .
eVars	The eVar variables are used for building custom conversion reports.

Element	Description
	See eVarN .
Props	Property (prop) variables are used for building custom traffic reports. See propN .
Dynamic Variable Prefix	A special prefix to the start of the value. The default prefix is "D=". See Dynamic Variables .

Page Views and Content

Field descriptions in Dynamic Tag Management for page views and content settings when deploying Analytics.

Property >  **Edit Tool** > **Page Views & Content**

Element	Description
Page Name	The name of each page on your site. See pageName .
Page URL Override	Overrides the actual URL of the page. See pageURL .
Channel (Site Section)	Identifies a section of your site. See channel .
Hierarchy	Determines the location of a page in your site's hierarchy. See hierN .

Link Tracking

Field descriptions in Dynamic Tag Management for link tracking when deploying Analytics.


Property >  **Edit Tool** > **Link Tracking**

Element	Description
Enable ClickMap	Determines whether visitor click map data is gathered. See s.trackInlineStats .
Track download links	Tracks links to downloadable files on your site. See s.trackDownloadLinks .

Element	Description
Download Extensions	<p>If your site contains links to files with any of the listed extensions, the URLs of these links will appear in reporting.</p> <p>See s.linkDownloadFileTypes.</p>
Track outbound links	<p>Determines whether any link clicked is an exit link.</p> <p>See s.trackExternalLinks.</p> <p>Single-Page App Considerations: Because of the way some SPA websites are coded, an internal link to a page on the SPA site might look like it is an outbound link.</p> <p>You can use one of the following methods to track outbound links from SPA sites:</p> <ul style="list-style-type: none"> • If you do not want to track any outbound links from your SPA, insert an entry into the Never Track section. <p>For example, <code>http://testsite.com/spa/#</code></p> <p>All # links to this host are ignored. All outbound links to other hosts are tracked, such as <code>http://www.google.com</code>.</p> <ul style="list-style-type: none"> • If there are some links that you want to track on your SPA, use the Always Track section. <p>For example, if you have a <code>spa/#/about</code> page, you could put "about" in the Always Track section.</p> <p>The "about" page is the only outbound link that is tracked. Any other links on the page (for example, <code>http://www.google.com</code>) are not tracked.</p> <p>Note that these two options are mutually exclusive.</p>
Keep URL Parameters	<p>Preserves query strings.</p> <p>See s.linkLeaveQueryString.</p>

Referrers and Campaigns

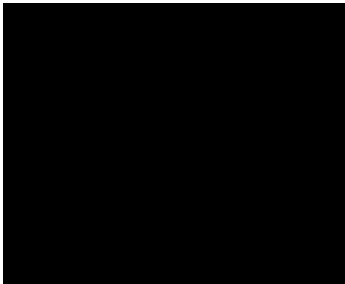
Field descriptions in Dynamic Tag Management for referrers and campaign options when deploying Dynamic Tag Management in Adobe Analytics.

Property >  **Edit Tool** > **Referrers & Campaigns**

Element	Description
Referrer Override	<p>Overrides the value set in the <code>s.referrer</code> variable, which is typically populated by the referrer set in the browser.</p> <p>See referrer.</p>

Element	Description
Campaign	A variable that identifies marketing campaigns used to bring visitors to your site. The value of campaign is usually taken from a query string parameter. See campaign .

Use the DTM interface to choose whether you want to use a Query String or Value (which could pull from a data element):



You can either enter your query string directly in the interface, or you can reference a separate data element if you have other means of tracking a campaign.

Cookies

Field descriptions for the Cookies global settings used for deploying Dynamic Tag Management in Adobe Analytics.

Property >  **Edit Tool** > **Cookies**


Element	Description
Visitor ID	Unique value that represents a customer in both the online and offline systems. See visitorID .
Visitor Namespace	Variable to identify the domain with which cookies are set. See visitorNamespace .
Domain Periods	The domain on which the Analytics cookie <code>s_cc</code> and <code>s_sq</code> are set by determining the number of periods in the domain of the page URL. This variable is also used by some plug-ins in determining the correct domain to set the plug-in's cookie. See s.cookieDomainPeriods .
FP Domain Periods	The <code>fpCookieDomainPeriods</code> variable is for cookies set by JavaScript (<code>s_sq</code> , <code>s_cc</code> , plug-ins) that are inherently first-party cookies, even if your implementation uses the third-party <code>2o7.net</code> or <code>omtrdc.net</code> domains. See s.fpCookieDomainPeriods .
Transaction ID	Unique value that represents an online transaction that resulted in offline activity.

Element	Description
	See transactionID .
Cookie Lifetime	Determines the life span of a cookie. See s.cookieLifetime .

Customize Page Code

Use field descriptions in Dynamic Tag Management to customize page code when deploying Analytics.

Add plugins to ensure that the code runs at the same time as the Analytics tool. For more information about the Analytics plugins, see [Implementation Plug-ins](#).

Property >  **Edit Tool** > **Customize Page Code**

Element	Description
Open Editor	You can insert any JavaScript call that must be triggered before the final <code>s.t()</code> call, which is contained in the <code>s_code</code> .
Execute	Before UI settings: Interface settings take precedence over the custom code (for example, if you want to override an eVar if a setting in the interface was enabled). After UI settings: Custom code takes precedence over interface settings.

Frequently Asked Questions About the Adobe Analytics Tool

A FAQ about the automatic configuration of the Adobe Analytics deployment. The automatic configuration method manages the AppMeasurement code for you.

Question	Answer
Where do I put my plugins when implementing Adobe Analytics via DTM?	If using DTM to manually host the <code>s_code</code> , plugins can be added in the same editor as the hosted <code>s_code</code> , just as it would be in a typical Adobe Analytics implementation. However, it is also an option to place the plugins in the editor within the <i>Customize Page Code</i> section of the tool settings. Both implementation methods should be equally effective. See DTM Switch Plugins .
If I make configuration changes in the new version of the tool, can I test in staging before publishing to production?	Yes. All changes can be tested in staging just like you normally would before deploying to a production environment. If you choose not to publish, because you notice issues in staging, the production code will continue to function as it did before the new integration was released.

Question	Answer
If I switch from manual configuration (the default setting for existing tools) to automatic configuration, will my current settings be affected?	No.
If I switch from manual library management to Managed by Adobe, will my current settings or code be affected?	Any user code that you have specified is overwritten with the base AppMeasurement library. You must move this code to the new Custom Page Code section at the end of the tool configuration so that the code continues executing. This method allows the AppMeasurement library to be managed (and upgraded) separately from the user's custom code.
Will the revision history for the Adobe Analytics tool be retained when the new integration is released?	Yes.

See [Add Adobe Analytics Tool](#) for configuration information.

Potential Pitfalls

There is a small chance that the integration could cause data collection issues if you currently use Adobe Analytics. These issues could arise only if you publish your library to production subsequent to the release. (Production code remains intact until publishing occurs.)

To avoid these issues, ensure that:



- Report suite IDs are correctly entered in the tool.
- Report suite IDs in the tool match the IDs in the AppMeasurement code.
- The currency code, character set, tracking server, and SSL tracking server configuration fields are correctly set with supported values.
- Custom code is defined in Library Management.

Create a Data Element

Create a data element in Dynamic Tag Management.

1. [Create Web Property](#), if you haven't done so already.
2. In the web property, click **Rules > Data Elements**.
3. Click **Create New Data Element**.
4. Complete the following fields and options:

Option	Description
Name	The data element friendly name that a marketer can recognize. For example, Product ID.

Option	Description
	 Note: The name is referenced by the rules builder, not an ID. If you change the name of the Data Element, you must change its reference in every rule that uses it.
Type	<p>Specifies where the data is pulled from, such as JS Object, CSS Selector, Cookie, URL Parameter, or Custom Script.</p> <p>Depending on which type you select, different options display. See Types of Data Elements in the Dynamic Tag Management Product Documentation for more information and examples.</p>
Default Value	<p>A default element. This value ensures that the data element always has a value, even if a URL parameter does not exist or cannot be found by Dynamic Tag Management.</p>  Note: If there is no value and no default value, then nothing is returned. Any variable referencing that data element won't get set. Note also that the default value field is ignored if it's a "custom code" data element.
Force lowercase value	Dynamic Tag Management automatically makes the value lowercased.
Remember this value for	<p>How long you want Dynamic Tag Management to remember this value.</p> <p>Valid values include:</p> <ul style="list-style-type: none">• Session: Session-based timing can vary depending on the implementation. Session data elements are set to the session cookie. However, this setting could be based on a web server or the browser. It is not related to the session used in marketing reports & analytics.• Pageview• Visitor

See [Data Elements](#) in the Adobe Tag Management Product Documentation for more information about how to use data elements.

5. Click **Save Data Element**.

Manually implement Adobe Analytics (legacy)

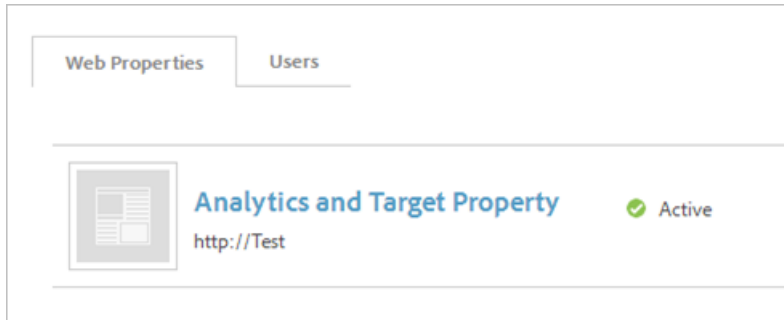
Create an Adobe Analytics tool for deployment using Dynamic Tag Management. This procedure describes a manual (legacy) implementation.

For information about automatic implementation management, see [Add Adobe Analytics Tool](#).

If you want to change a manual configuration to automatic, edit a tool and click **Enable Automatic Configuration**.

1. Download Analytics measurement code:
 - a) In Adobe Experience Cloud, click **Reports & Analytics > Admin Tools**.

- b) Click **Code Manager**.
 - c) Click [JavaScript \(new\)](#) to download the code locally.
2. In Dynamic Tag Management, [create a web property](#).



After you create the web property, it is available for editing on the **Web Properties** tab on the **Dashboard**. Activating the web property is not required

3. Add an Adobe Analytics tool to the property:
- a) On the **Web Properties** tab, click the property.
 - b) On the **Overview** tab, click **Add a Tool**.
 - c) From the **Tool Type** menu, select **Adobe Analytics**.

Add a Tool

Tool Type

Tool Name

Configuration method
 ?

Authenticate via:
 Marketing Cloud Web Services ?

Web Services Username

Shared Secret

d) Configure the following fields:

Element	Description
Tool Type	The Experience Cloud solution, such as Analytics, Target, Social, and so on.
Tool Name	The name for this tool. This name displays on the Overview tab under Installed Tools .

Element	Description
Production Account ID	A number for your production account for data collection. Dynamic Tag Management automatically installs the correct account in the production and staging environment.
Staging Account ID	A number used in your development or test environment. A staging account keeps your testing data separate from production.

- Click **Create Tool**.

The installed tool displays on the **Overview** tab.



- To configure the code, click **Settings** (⚙️).

At a minimum, click **Cookies** and configure your tracking server and SSL tracking server.

- Click **General** and *insert the core AppMeasurement code*.
- Define a *page load rule* to collect Analytics data.

You are now ready to define rules to collect analytics data. You might want to define a few data elements first. Data elements let you extract data from the page that you can use to configure your rule. To get started, you can define a page load rule that does not have any conditions to collect Analytics data on each page.

- Add the header and footer code* on the Embed tab.

For staging, you can leave the default Amazon hosting option. You can change it if needed before your production rollout.

- (Optional) Click **Settings** (⚙️) on the Options tab, and configure the **Adobe Analytics** code.



Note: The settings on the **Adobe Analytics** page (General, Cookies, and so on) override settings in your `s_code`. If these settings exist in your `s_code`, there is no need to reiterate them here.

Create New Rule

Steps that describe how to create rules in Dynamic Tag Management.

- Create Web Property*, if you haven't done so already.
- In the web property, click the **Rules** tab.
- Select the type of rule you wish to create from the left navigation pane, such as Event Based Rules or Page Load Rules.
- Click **Create New Rule**.
- Name the rule and select a category, if you wish.
- Next, set up the condition(s) for the rule. The setup differs depending on the type of rule you are implementing.

Type of Rule

Refer to this Topic

Event-based rule

[Create Conditions for Event-Based Rules](#)

Page Load rule

[Create Conditions for Page-Load Rules](#)

Type of Rule**Refer to this Topic****Direct Call rule**[Create Conditions for Direct-Call Rules](#)

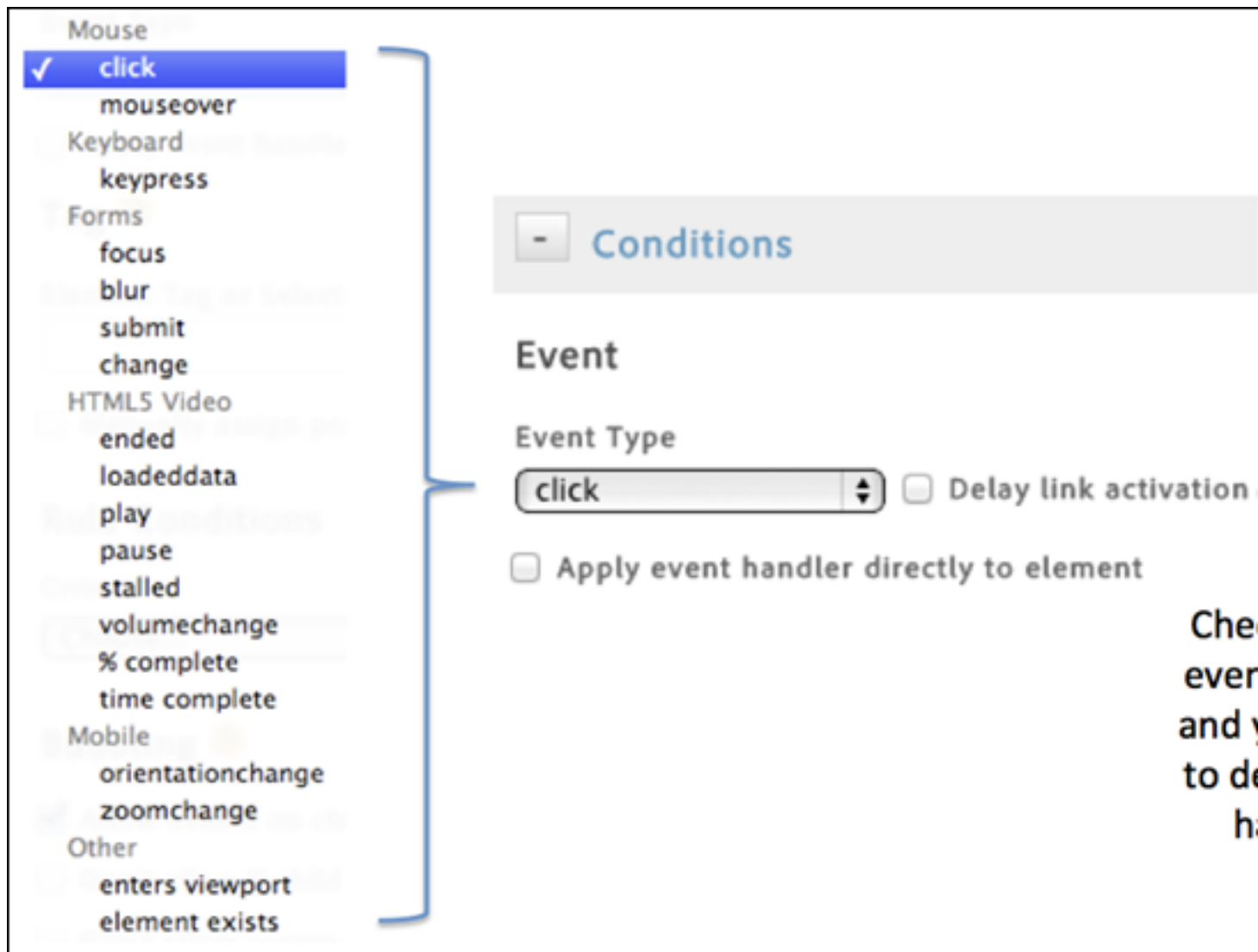
The category field is only for your own organizational purposes and is not required. You can delete categories by clicking the x icon in the category.

7. [Set Up Actions for the Condition to Trigger.](#)

Create Conditions for Event-Based Rules

Conditions determine when an event-based rule is triggered.

1. Select the type of interaction you want to track, such as mouse clicks, or submitting a form.



For more information, see [Event Types](#) in the Adobe Tag Management Product Documentation.

2. Enable the following options as necessary:

Element	Description
Delay Link Activation	Enable if the event activates a link and you want the link to delay until the event has time to fire.
Apply event handler directly to element	Applies the event handler to the specific element that is targeted. This setting is tied to the bubbling and layering concept in a browser.

For example, when you click an image inside an anchor tag like ``, you might expect the click to be associated with the anchor tag, because the tag is in the bubble stream. However, when you inspect the click in the developer tools, the click may actually affect only the `` tag. To ensure that the event is handled correctly, associate the click with the `` tag and do not depend on the browser to bubble up the click to a parent element. An event like a click can potentially bubble up to `<body>`. It is important to understand where the event is actually bound, and target it specifically to make sure that the rule fires correctly.

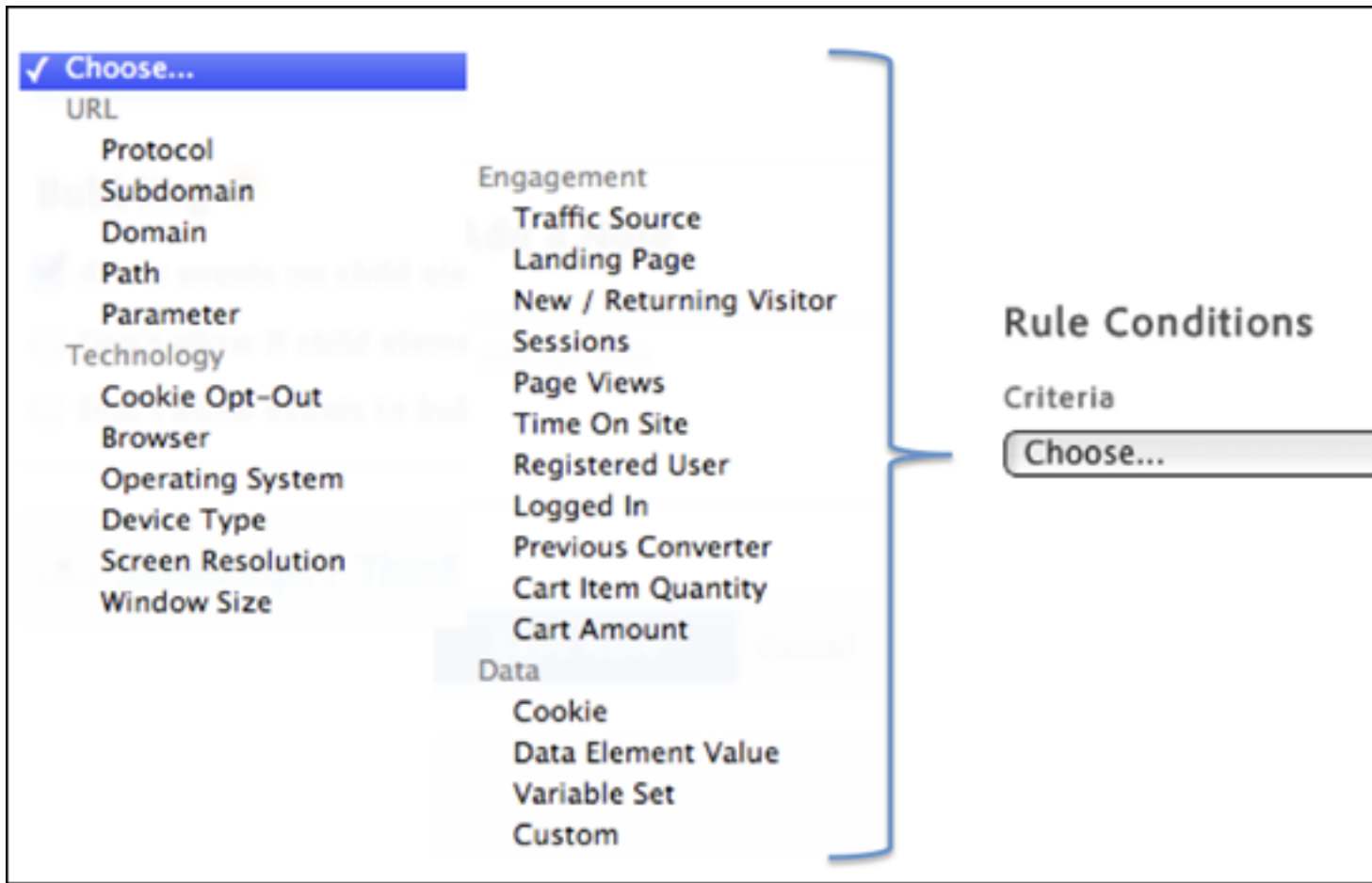
Bubbling means that the event is first captured and handled by the inner most element and then propagated to outer elements.

3. Indicate the name of the tag you want to track, and additional properties the tag has that you want to match.

The screenshot shows a configuration interface for a dynamic tag. At the top, there is a 'Tag' label with a question mark icon. Below it is a text input field labeled 'Element Tag or Selector' containing the text 'a.contact'. To the right of this field, an arrow points to a text annotation: 'You can identify additional properties in the selector box by leveraging CSS selector syntax'. Below the selector field is a checked checkbox labeled 'Manually assign properties & attributes'. To the right of this checkbox, an arrow points to another text annotation: 'OR you can check this but additional properties manually'. Below the checkbox, there are two input fields: 'Property' containing 'class' and 'Value' containing 'contact', separated by an equals sign (=).

See [Using the CSS Selector](#) in the Dynamic Tag Management Product Documentation for information about finding the correct element tag.

4. Select and set up any additional criteria or condition types you wish to bind to the rule.



5. Indicate your preference regarding event bubbling.

Event bubbling is one way of event propagation in HTML DOM.

If you...

Check this option

Want related interactions on child elements of the rule selector you identified to fire the rule.

Allow events on child elements to bubble.

Want to prevent bubbling when the child element already triggers its own event.

Do not allow if child element already triggers event.

Don't want the events of the rule selector you identified to go beyond the element itself in the event hierarchy.

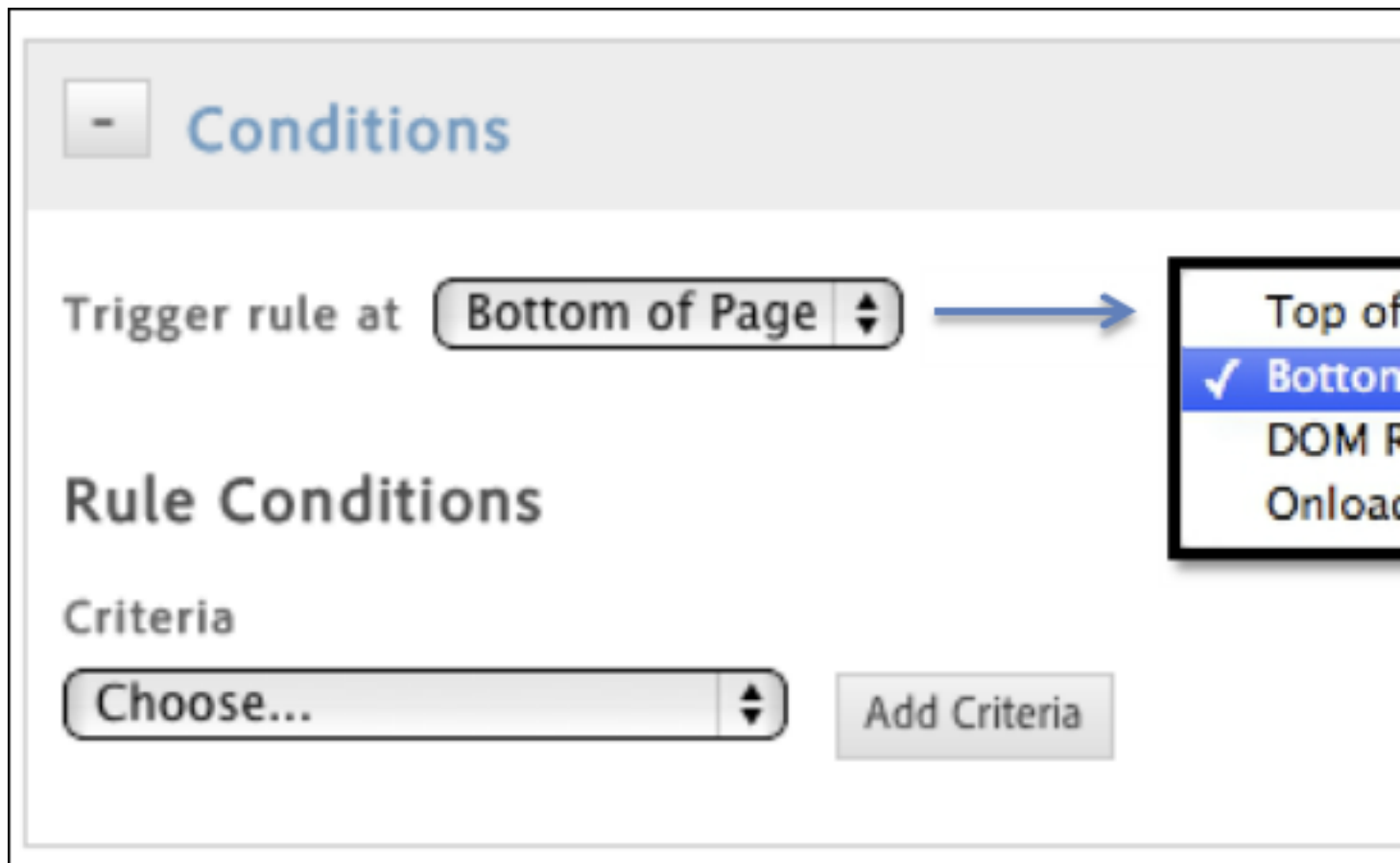
Do not allow events to bubble upwards to parents.

Create Conditions for Page-Load Rules

Create rules that determine on what pages a rule is triggered.

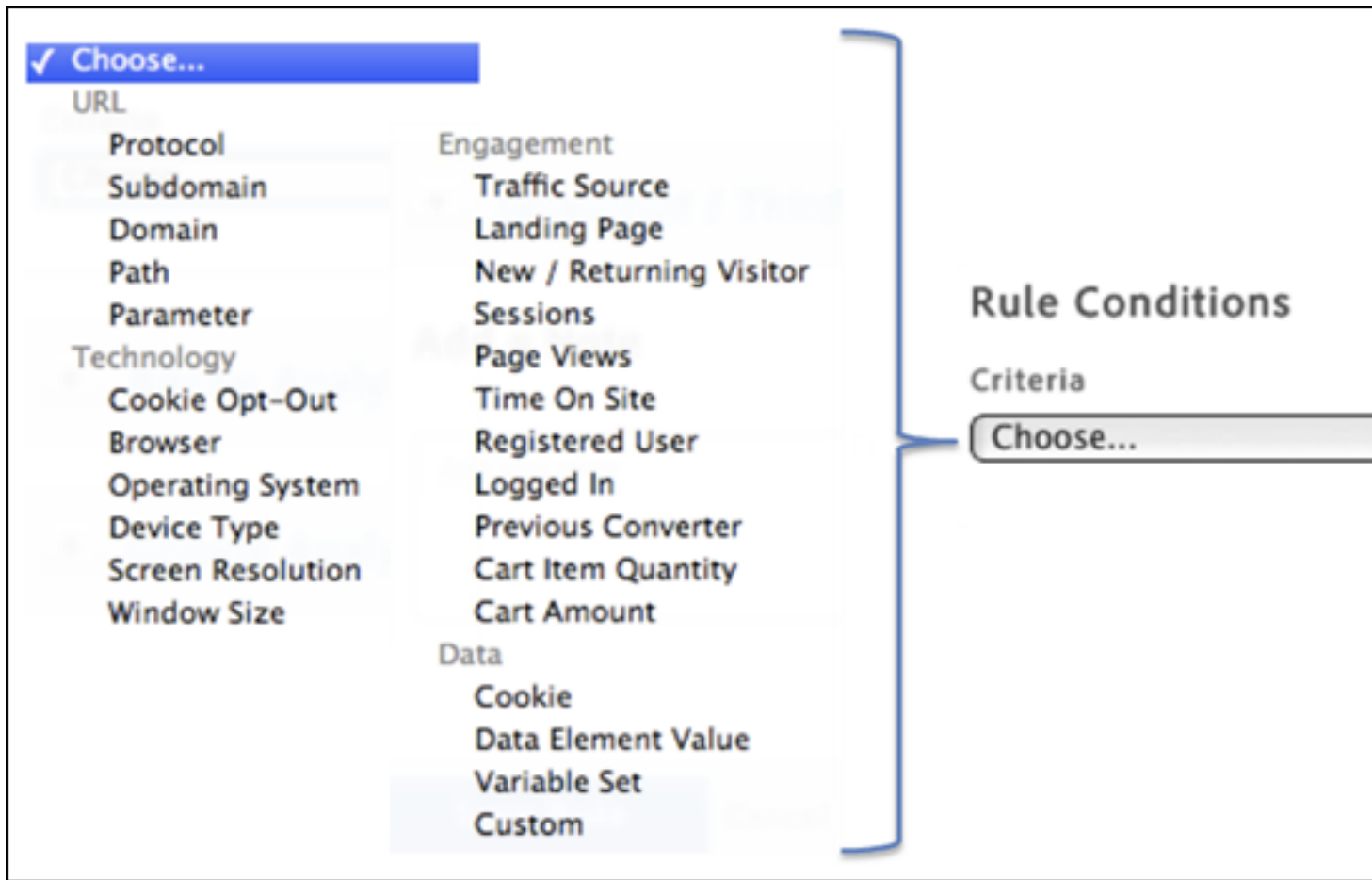
1. Specify where on the page you want the rule to trigger.

The timing of where the rule fires on the page becomes more important when there are dependencies on page content within the rule.



2. Specify the condition that causes the rule to fire.

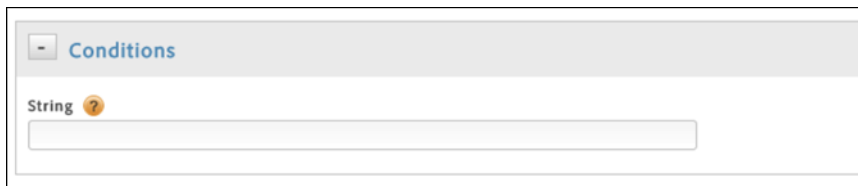
For example, you can select **Path** to identify specific pages for which you want the rule to fire.



Create Conditions for Direct-Call Rules

Create conditions for direct-call rules.

In the **Conditions** dialog, specify the string that will be passed to `_satellite.track()` in your direct call, without quotes.



Note: If you specify the string that will be passed to `_satellite.track()` in your direct call using the UI, as described above, do not use quotation marks. If you insert [customized page code](#) using the editor, you must use quotation marks.


Set Up Actions for the Condition to Trigger

Set up actions that you want the condition to trigger.

After setting up the condition, you must set up the actions that you want the condition to trigger. These actions can include Analytics events, third-party tags, and custom scripts. This example describes how to set up scripts or third-party tags.

Beyond integrated tools like Adobe Analytics and Google Analytics, Dynamic Tag Management can trigger any type of JavaScript or inject HTML into your site, in select pages or in specific scenarios.

Each rule can trigger as many scripts or HTML injections as you want.

 **Note:** Because DTM allows you to inject custom code into your page, please take care not to create cross-site scripting (XSS) vulnerabilities (see [OWASP's guide](#) for more info). Using data elements within a script requires particular attention. Always assume data element values might come from an untrusted source.

To set up actions for the condition to trigger

1. Click **JavaScript / Third Party Tags** to add a new script to your rule.



2. Click **Add New Script**.



3. Name the script.

4. Specify how you want the script to trigger, and paste the desired content into the text

New Javascript / Third Party Script

Tag Name

Type

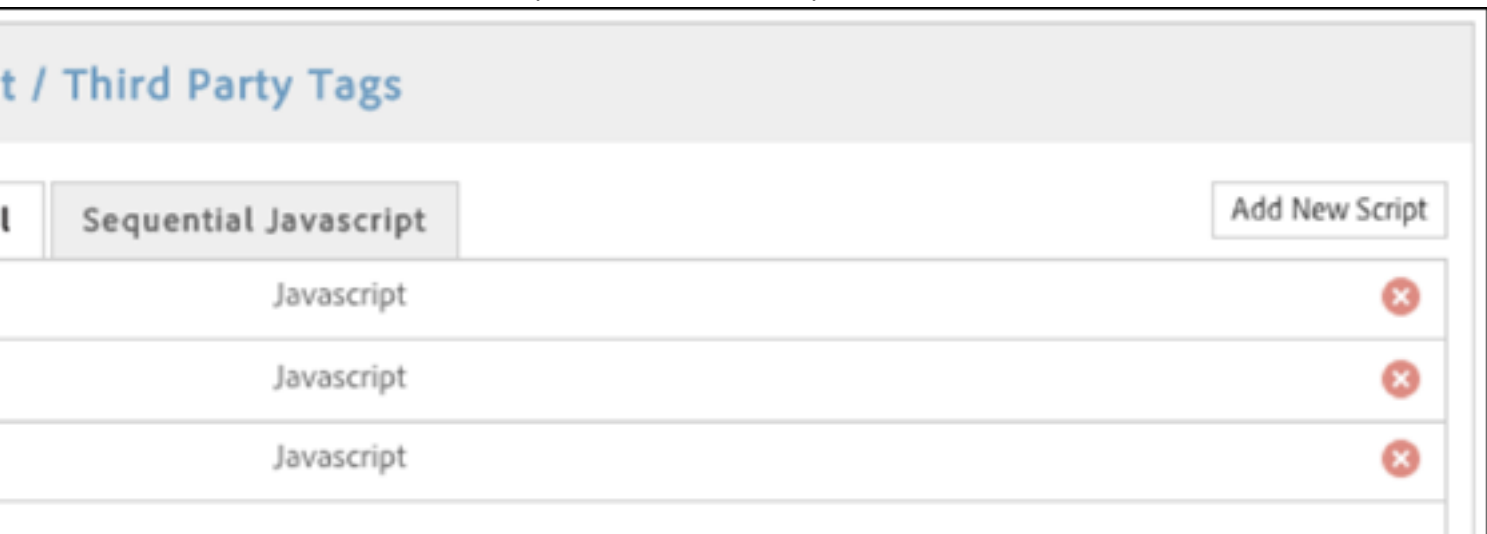
Non-Sequential Javascript ⌵

Execute Globally ?

```
1
```

Save Code Cancel

5. Click **Save Code**, and the script will be added to the queue for the



Test Unpublished Rules for Akamai Hosting

Test unpublished rules from your console if you use Akamai hosting.

The Switcher plugin is often the easiest way to test. See [Search Discovery Plugins](#) in the Dynamic Tag Management Product Documentation for more information.

The following steps show how to test without using the Switcher plugin:

1. Access your web console on your site and type `localStorage.setItem('sdsat_stagingLibrary', true)`.
2. Press **Enter**.
3. Type `_satellite.setDebug(true)`, then press **Enter**.
4. Refresh the page.

This action loads your staging library and sets the debugger, so that you can see details of all available (published / unpublished) rules firing on the page.

5. When finished, run `localStorage.setItem('sdsat_stagingLibrary', false)`, then press **Enter**.

Test Rules for Library Download or FTP

If you use library download or FTP delivery, or you do not have a testing environment, you can use a Rewrite tool such as Charles to test unpublished rules. This need varies based on your specific implementation.

Implement Analytics using JavaScript

To begin using Analytics, data must be sent to a report suite to display in reporting.

The easiest and recommended way to send data to Analytics is by using [Dynamic Tag Management](#). However, in some cases, you might want to implement Analytics using the older JavaScript method.



Note: This section describes the legacy method of implementing Analytics. All Analytics customers have access to [Dynamic Tag Management](#) which is the standard method to deploy Experience Cloud tags.

Implementation Steps

To successfully implement a page with code to collect data, you must have access to your hosting servers to upload new content to your website. It is also useful to have an existing site to implement code.

The following steps walk you through a basic Analytics implementation.

Step	Task	Description
1	Download AppMeasurement for JavaScript and the Visitor ID service.	The download is available in Code Manager . This download zip contains several files. <code>AppMeasurement.js</code> and <code>VisitorAPI.js</code> are the relevant files when implementing Analytics.
2	Set up the Experience Cloud ID service.	(Formerly <i>Visitor ID service</i> .) See Set Up the Experience Cloud ID Service for Analytics .
3	Update <code>AppMeasurement.js</code> .	Copy the Example AppMeasurement.js Code and paste it at the beginning of your <code>AppMeasurement.js</code> file. At a minimum, update the following variables: <ul style="list-style-type: none"> <code>s.account="INSERT-RSID-HERE"</code> <code>s.trackingServer="INSERT-TRACKING-SERVER-HERE"</code> <code>s.visitorNamespace="INSERT-NAMESPACE-HERE"</code> <code>s.visitor=Visitor.getInstance("INSERT-MCORG-ID-HERE")</code> See Correctly populate the trackingServer and trackingServerSecure variable or contact Client Care if you are unsure about any of these values. If they are not set correctly, data will not be collected by your implementation.
4	Host <code>AppMeasurement.js</code> and <code>VisitorAPI.js</code> .	These core JavaScript files must be hosted on a web server that is accessible to all pages on your site. You need the path to these files in the next step.
5	Reference <code>AppMeasurement.js</code> and <code>VisitorAPI.js</code> on all site pages.	Include the Visitor ID Service by adding the following line of code in the <code><head></code> or <code><body></code> tag on each page. <code>VisitorAPI.js</code> must be included before <code>AppMeasurement.js</code> : <pre><script language="JavaScript" type="text/javascript" src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/VisitorAPI.js"></script></pre> Include AppMeasurement for JavaScript by adding the following line of code in the <code><head></code> or <code><body></code> tag on each page: <pre><script language="JavaScript" type="text/javascript" src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/AppMeasurement.js"></script></pre>

Step	Task	Description
6	Update and deploy page code.	Copy the Example Page Code and paste it just after the opening <code><body></code> tag on each page you want to track. At a minimum, update the following variables: <ul style="list-style-type: none"> <code>var s=s_gi("INSERT-RSID-HERE")</code> <code>s.pageName="INSERT-NAME-HERE" // for example,</code> <code>s.pageName=document.title</code>
7	Use the DigitalPulse Debugger to verify that data is being sent.	Install the bookmarklet. After it is installed, load a page where you have deployed page code and then open the debugger. The debugger displays details about the collection data that was sent.

Caching

The JavaScript file is cached in the visitor's browser after it is initially loaded, and is typically downloaded no more than once per session. The file is not downloaded on each page, even though it is used by every page on the site. On most websites, users average more than a few page views per session, so transferring JavaScript that is used multiple times into this file can result in less overall downloaded data.

JavaScript for AppMeasurement Compression

If you are concerned about page weight (size) of H Code (AppMeasurement for JavaScript 1.0 is pre-compressed), Adobe recommends that you consider compressing the file using GZIP. GZIP is supported by all major browsers and offers better performance than JavaScript compression to compress and decompress the core `s_code.js` JavaScript file.


The following links help explain how you can use GZIP functionality on your site to handle compression of the `s_code.js` JavaScript code:

[Apache Module mod_deflate](#)


[Enabling gzip compression with Tomcat and Flex](#)

Example Page Code and Global Configuration

This section contains example code for your core JavaScript file and the pages on your site.

 **Important:** This example uses the visitor ID service, which is deployed as part of your [Implement Analytics using JavaScript](#). Enabling the visitor ID service in AppMeasurement before you have included the Visitor API JavaScript file on all site pages could result in duplicate visitor counts. To avoid duplicate visitor counts, make sure that you understand and follow the process described in [Visitor ID Service](#).

Example AppMeasurement.js Code

 **Important:** Configuration variables should be set above the `doPlugins` function.

For new implementations, you can paste the following global configuration code at the beginning of `AppMeasurement.js` to get started:

```
//initialize AppMeasurement
var s_account="INSERT-RSID-HERE"
```

```

var s=s_gi(s_account)

/***** VISITOR ID SERVICE CONFIG - REQUIRES VisitorAPI.js *****/
s.visitor=Visitor.getInstance("INSERT-MCORG-ID-HERE")

/***** CONFIG SECTION *****/
/* You may add or alter any code config here. */
/* Link Tracking Config */
s.trackDownloadLinks=true
s.trackExternalLinks=true
s.trackInlineStats=true
s.linkDownloadFileTypes="exe,zip,wav,mp3,mov,mpg,avi,wmv,pdf,doc,docx,xls,xlsx,ppt,pptx"
s.linkInternalFilters="javascript:" //optional: add your internal domain here
s.linkLeaveQueryString=false
s.linkTrackVars="None"
s.linkTrackEvents="None"

/* uncomment below to use doPlugins */
/* s.usePlugins=true
function s_doPlugins(s) {

// use implementation plug-ins that are defined below
// in this section. For example, if you copied the append
// list plug-in code below, you could call:
// s.events=s.apl(s.events,"event1","",1);

}
s.doPlugins=s_doPlugins */

/* WARNING: Changing any of the below variables will cause drastic
changes to how your visitor data is collected. Changes should only be
made when instructed to do so by your account manager.*/
s.trackingServer="INSERT-TRACKING-SERVER-HERE"
s.trackingServerSecure="INSERT-SECURE-TRACKING-SERVER-HERE"

/***** PLUGINS SECTION *****/

// copy and paste implementation plug-ins here - See "Implementation Plug-ins" @
// https://marketing.adobe.com/resources/help/en_US/sc/implement/#Implementation_Plugins
// Plug-ins can then be used in the s_doPlugins(s) function above

/***** MODULES *****/

// copy and paste implementation modules (Media, Integrate) here
// AppMeasurement_Module_Media.js - Media Module, included in AppMeasurement zip
// AppMeasurement_Module_Integrate.js - Integrate Module, included in AppMeasurement zip

/* ===== DO NOT ALTER ANYTHING BELOW THIS LINE ! =====

```

Example Page Code

For new implementations, you can paste the following page code just after the opening <body> tag on pages you want to track:

```

<script language="JavaScript" type="text/javascript"><!--
/* You may give each page an identifying name, server, and channel on
the next lines. */
s.pageName=""
s.server=""
s.channel=""
s.pageType=""
s.prop1=""
s.prop2=""
s.prop3=""
s.prop4=""
s.prop5=""
/* Conversion Variables */
s.campaign=""
s.state=""

```

```
s.zip=""
s.events=""
s.products=""
s.purchaseID=""
s.eVar1=""
s.eVar2=""
s.eVar3=""
s.eVar4=""
s.eVar5=""
var s_code=s.t();if(s_code)document.write(s_code)//--></script>
```

Make sure that you have also included a reference to `AppMeasurement.js` and `VisitorAPI.js` on each page. See [Implement Analytics using JavaScript](#) for instructions.

About AppMeasurement for JavaScript

AppMeasurement for JavaScript is a new library that provides the same core functionality of `s_code.js`, but is lighter and faster for use on both mobile and desktop sites.

Things to know before you migrate

The following list contains changes you need to understand before switching to this new AppMeasurement version:

- Some plug-ins are no longer supported. See [AppMeasurement Plug-in Support](#).
- The library does not support dynamic account selection ([s.dynamicAccountList](#), [s.dynamicAccountMatch](#), and [s.dynamicAccountSelection](#)).
- The library and page code can be deployed inside the `<head>` tag.
- The Media and Integrate modules are supported using updated module code that is in the JavaScript AppMeasurement download package. The Survey module is not supported.
- Your existing page code is compatible with the new version.
- The library provides native utilities to get query parameters, read and write cookies, and perform advanced link tracking.

Frequently Asked Questions

See the [FAQ](#) for information about performance, video tracking, mobile, and more.

Initialization process

Call `s_gi()`, passing the report suite ID to initialize an AppMeasurement instance:

```
var s_account="INSERT-RSID-HERE"
var s=s_gi(s_account)
```

When `s_gi` is called, if an AppMeasurement instance does not exist for the specified `s_account`, a new instance is created. Otherwise the existing instance is returned. This helps avoid creating duplicate objects for the same account.

Retrieve an AppMeasurement instance

Throughout your code, call the global `s_gi()` function to retrieve an existing AppMeasurement instance.

Utilities

JavaScript AppMeasurement provides the following built-in utilities:

- [Util.cookieRead](#)
- [Util.cookieWrite](#)
- [Util.getQueryParam](#)

Clear Vars

A new `clearVars` method is available to clear the following values from the instance object:

- `props`
- `eVars`
- `hier`
- `list`
- `events`
- `eventList`
- `products`
- `productsList`
- `channel`
- `purchaseID`
- `transactionID`
- `state`
- `zip`
- `campaign`

For example:

```
s.clearVars()
```

Benefits

- 3-7x faster than H.25 code.
- Only 21k uncompressed and 8k gzipped (H.25 code is 33k uncompressed and 13k gzipped).
- Native support for several common plugins ().
- Small and fast enough to be used with mobile sites, and robust enough to be used on the full desktop web, allowing you to leverage a single library across all web environments.

Migrating to AppMeasurement for JavaScript

The following table contains a list of tasks you need to perform to migrate your implementation.



Note: We recommend migrating to the [Experience Cloud ID Service](#) when you migrate to `AppMeasurement` for JavaScript.

Step	Task	Where	Description
1	Check plug-in compatibility	<code>s_code.js</code>	Some plug-ins are longer supported. See AppMeasurement Plug-in Support .
2	Download the new AppMeasurement	Admin Console > Code Manager	The download zip contains a minified <code>AppMeasurement.js</code> file, and Javascript files for the Media and Integrate modules.
3	Copy your <code>s_code.js</code> customization to <code>AppMeasurement.js</code> .	<code>s_code.js</code> and <code>AppMeasurement.js</code>	Move all code that appears before the <code>DO NOT ALTER ANYTHING BELOW THIS LINE</code> section in <code>s_code.js</code> to the beginning of <code>AppMeasurement.js</code> .

Step	Task	Where	Description
4	(Optional) Update plug-ins	<code>AppMeasurement.js</code>	If you are using the <code>getQueryParam</code> plug-in, update these calls to use the new utility, Util.getQueryParam .
5	(Optional) Update Media and Integrate modules	<code>AppMeasurement.js</code>	If you are using either of these modules, copy and paste the code from <code>AppMeasurement_Module_Media.js</code> and/or <code>AppMeasurement_Module_Integrate.js</code> and paste it just before the <code>DO NOT ALTER ANYTHING BELOW THIS LINE</code> in <code>AppMeasurement.js</code> .
6	Deploy new JavaScript	Your website	The new JavaScript file can be deployed according to your standard process. Your existing page code is compatible with this version.

AppMeasurement Plug-in Support

Plug-in support in the current version of JavaScript AppMeasurement.

Tested Plug-ins

The following plug-ins were tested and verified as compatible:

- [s.abort flag](#)
- [appendList](#)
- [doPlugins Function](#)
- [getAndPersistValue](#)
- [getDaysSinceLastVisit](#)
- [getLoadTime](#)
- [getNewRepeat](#)
- [getPageVisibility](#)
- [getPercentPageViewed](#)
- [getPreviousValue](#)
- [getQueryParam](#)
- [getTimeParting](#)
- [getValOnce](#)
- [getVisitNum](#)
- [getVisitStart](#)
- [hitGovernor](#)
- [Internal Traffic](#)
- [performanceTiming](#)
- [trackTNT](#)

Untested Plug-ins

The following plug-ins should continue to work since the underlying functionality is still supported, but they have not been tested and verified as compatible. You should test these plug-ins in your development environment before migration.

- `getActionDepth`
- `getCookiesAccepted`

Accelerated Mobile Pages

Implement the Accelerated Mobile Pages (AMP) project in Adobe Analytics.

AMP is an [open source project](#) that lets you build web pages for static content that renders quickly. This feature is ideal for publishers who want to create mobile-optimized content once, and have it load instantly everywhere. Topics include:

- [How it Works](#)
- [Using the amp-analytics tag with the "adobeanalytics" template](#)
- [Using the amp-analytics tag with the "adobeanalytics_nativeConfig" template](#)
- [Summary](#)
- [Frequently Asked Questions](#)

Additional Documentation and Examples

- External, open-source AMP project [documentation](#)
- Setup [examples](#)

How it Works

AMPs have specially tagged HTML pages cached around the web on different content delivery networks (CDNs) of participating technology partners and publishers. By doing this, AMP content is delivered from the closest possible source with the lowest possible latency. This creates an analytics challenge because you can never be 100% sure where a publisher's content will be loaded from, and third-party cookies are troublesome for visitor identification.

In addition, to dramatically reduce page weight and speed page load time, AMPs restrict the use of JavaScript and cookies. While this is advantageous for your mobile device because it reduces the amount of processing, it introduces challenges to accurately measuring unique visitors and understanding user acquisition and retention.

To solve these problems, Adobe has collaborated with AMP partners and publishers on two options that a publisher can choose from to best suit their business needs, both using the `amp-analytics` tag. The first approach uses the "adobeanalytics" tracking template to construct the Analytics request directly from within the AMP. The second approach uses the "analytics_nativeConfig" tracking template, which uses an iframe containing the AppMeasurement code you deploy on your normal site. The following table gives you an idea of the pros and cons of each approach.

	"adobeanalytics" template	"adobeanalytics_nativeConfig" template
Visitor/visit counts (in existing report suite)	High inflation	Minimal inflation
Using a separate report suite	Recommended	Not necessary
New vs. return visitors	Not supported	Supported
Visitor ID service	Not supported	Supported

Video & link tracking	Partial support	Not yet supported
Difficulty of implementation	Somewhat difficult	Relatively easy
Experience Cloud integrations	Not supported	Supported with caveats

Using the amp-analytics tag with the "adobeanalytics" template

The "adobeanalytics" tracking template utilizes the `amp-analytics` tag to construct a tracking request directly. Using the "adobeanalytics" template in the `amp-analytics` tag, you can specify hit requests that fire on specific page events, like the page becoming visible or on a click (and in the future, video views and more). Click events can be customized to apply to certain element IDs or classes by specifying a selector. Adobe has made this easy to set up using the "adobeanalytics" template specifically designed for Adobe Analytics. You can load the template by adding `type="adobeanalytics"` to the `amp-analytics` tag.

In the following code example, there are two triggers defined: `pageLoad` and `click`. The `pageLoad` trigger will fire when the document becomes visible and will include the `pageName` variable as defined in the `vars` section. The second trigger `click` will fire when a button is clicked. `eVar 1` will be set for this event with the value `button clicked`.

```
<amp-analytics type="adobeanalytics">
<script type="application/json">
{
  "requests": {
    "myClick": "${click}&v1=${eVar1}",
  },
  "vars": {
    "host": "metrics.example.com",
    "reportSuites": "reportSuiteID",
    "pageName": "Adobe Analytics Using amp-analytics tag"
  },
  "triggers": {
    "pageLoad": {
      "on": "visible",
      "request": "pageView"
    },
    "click": {
      "on": "click",
      "selector": "button",
      "request": "myClick",
      "vars": {
        "eVar1": "button clicked"
      }
    }
  }
}
</script>
</amp-analytics>
```

In the `click` trigger, you can specify a selector to ensure that whenever the specific DOM element is clicked (in this case, any button), the `buttonClick` request is fired and will be automatically set to denote this hit as a non-stage event (i.e. `trackLink` call).

Additionally, `amp-analytics` supports a number of variable substitutions so that AMP can provide data values that it is aware of. You can learn all about those and more by visiting: [amp-analytics variable documentation](#).

Be aware that if you want to incorporate any technology or DOM variables (such as browser, screen size, device, referrer, etc.) you will have to explicitly add them to any request, as they are not automatically generated for you. Documentation on each of our available query string parameters used for tracking can be found [here](#).

If you inspect the hits created by amp-analytics, you will notice that in each request, Adobe has included the `vid` query parameter. We set the `vid` based on a built-in AMP function to set a custom Analytics cookie ID named `adobe_amp_id`. This ID is independent of any other ID being set by Adobe Analytics elsewhere (e.g. `s_vi` cookie) and creates new visitors in any report suite the hits are sent to.

There are a few caveats to be aware of. When using the amp-analytics tag as mentioned above, visitors will be independent of your normal tracking, and because the AMP can be loaded from any content delivery network, you will get a unique visitor for each CDN a visitor sees this AMP on (hence the visitor inflation mentioned previously). For this reason, Adobe recommends that if you use the "adobeanalytics" template for amp-analytics, you put your data into a separate report suite specific to AMP. Also, the Experience Cloud ID service (formerly, *visitor ID service*) is not supported using this method, so if your business requires additional Experience Cloud integrations, or will in the future, this is probably not the option for you.

Finally, and perhaps most importantly, this amp-analytics solution requires that the tracking server you specify in the `vars` section matches the tracking server on your main site, so that your existing privacy policy controls are respected. Otherwise, you must create a separate privacy policy just for AMPs.

Using the amp-analytics tag with the "adobeanalytics_nativeConfig" template

The "adobeanalytics_nativeConfig" tag is easier to implement, as it will use the same tagging methodology you use on your normal web pages. To accomplish this, add the following to your amp-analytics tag:

```
<amp-analytics type="adobeanalytics_nativeConfig">
  <script type="application/json">
    {
      "requests": {
        "base": "https://${host}",
        "iframeMessage":
"${base}/stats.html?campaign=${queryParam(campaign)}&pageURL=${ampdocUrl}&ref=${documentReferrer}"
      },
      "vars": {
        "host": "statshost.publishersite.com"
      },
      "extraUrlParams": {
        "pageName": "Adobe Analytics Using amp-analytics tag",
        "v1": "eVar1 test value"
      }
    }
  </script>
</amp-analytics>
```

This approach sends data to a utility web page via special query string parameters added to the `iframeMessage` request parameter. In this case, notice that we have added the `ampdocUrl` AMP variable, and the `documentReferrer` to the query string parameters `pageURL`, and refer respectively to the `iframeMessage` request above. These extra query string parameters can be named whatever you like, as long as your `stats.html` page (shown below) is configured to collect the appropriate data from them.

The "adobeanalytics_nativeConfig" template also adds query string parameters based on the variables listed in the `extraUrlParams` section of the amp-analytics tag. In this case, you can see we have specified the `pageName` and `v1` parameters, which will be used by our `stats.html` page.

Be aware that you can only use a single amp-analytics template at a time and can not use the "adobeanalytics" template as well as the "adobeanalytics_nativeConfig" template on the same AMP. If you attempt to do so, you might see an error in the browser console, and you will erroneously inflate your visitor count.

```
<html>
<head>
```

```
<title>Stats Test</title>
<script language="JavaScript" type="text/javascript" src="VisitorAPI.js"></script>
<script language="JavaScript" type="text/javascript" src="AppMeasurement.js"></script>
<html>
<head>
<title>Stats Test</title>
<script language="JavaScript" type="text/javascript" src="VisitorAPI.js"></script>
<script language="JavaScript" type="text/javascript" src="AppMeasurement.js"></script>
</head>
<body>
<script>
var v_orgId = "1234567@PublisherOrg";
var s_account = "reportSuite";
var s_trackingServer = "metrics.publisher.com";
var s_visitorNamespace = "publisherNamespace";
var visitor = Visitor.getInstance(v_orgId);
visitor.trackingServer = s_trackingServer;
var s = s_gi(s_account);
s.account = s_account;
s.trackingServer = s_trackingServer;
s.visitorNamespace = s_visitorNamespace;
s.visitor = visitor;
s.pagename = s.Util.getQueryParam("pageName");
s.eVar1=s.Util.getQueryParam("v1");
s.campaign=s.Util.getQueryParam("campaign");
s.pageURL=s.Util.getQueryParam("pageURL");
s.referrer=s.Util.getQueryParam("ref");
s.t();
</script>
</body>
</html>
```

As shown above, you can use or link to your existing `VisitorAPI.js` and `AppMeasurement.js` (as in our example), or whatever your existing implementation uses, then add the correct configuration parameters. To capture the correct values into the correct variables, you can use the provided `s.Util.getQueryParam` function to grab the value(s) that you passed in from the `iframeMessage` URL and set the appropriate variables, just as you would on a typical page. If you use tag management software like Adobe's [Dynamic Tag Manager](#), the query string parameters should be straightforward to capture. In this case, `s.pageName` is set to the value we passed in the query string parameter `pageName`. Here, the page name would be set to *Adobe Analytics Example 2*.



Important: Due to restrictions on iframes in the AMP framework, your `stats.html` page must be hosted on a separate subdomain from the domain the AMP itself is hosted on. The AMP framework does not allow for iframes from the same subdomain that the AMP page itself exists on. For example, if your AMP is hosted on `amp.example.com`, be sure to host your `stats.html` page on a separate subdomain such as `ampmetrics.example.com` or something similar.

Because the utility page is hosted on your original site, no additional work is needed to support your existing privacy policy across all AMPs. This means that if an end user opts out of tracking on your primary site, they are also opted out of tracking on all your AMPs, with no additional steps required. Using this utility page also means that AMP can support Adobe's Experience Cloud ID service so that you can integrate the measurement captured on your AMPs with the rest of the Experience Cloud (for targeted advertising using Adobe Audience Manager for example).

To reiterate, if your organization is not yet using the Experience Cloud ID service (or has tag management software like Adobe's Dynamic Tag Manager), you can tag the `stats.html` page however you want. Use your existing implementation as a reference point. The only difference from your standard implementation is that you will get the applicable data points from the `amp-analytics iframeMessage` URL (or `document.URL` from within the `stats.html` page) for each of the variables you want to set. Also, if you want to use any of the [AMP specific variables](#) (as mentioned above) like the AMP referrer or AMP page URL, please include them in the `iframeMessage` object as shown in our example above.

As flexible as this solution is, there are caveats. Due to inherent restrictions in the `amp-analytics iframeMessage`, it can only be loaded on a page load once. This means you will not be able to do link tracking or video tracking with the `"adobeanalytics_nativeConfig"` template. Moreover, some DOM values that are typically captured automatically by our AppMeasurement code, such as `referrer` (which impacts the Search Engine Keyword reports, Referrer, and Referrer Type reports, or may include a marketing campaign tracking code) will have to be passed manually to the `iframeMessage` using whatever AMP variables [are available](#). For this reason, Adobe recommends setting a custom variable with the value AMP if you put AMP data into an existing report suite, so that you can segment out AMP traffic when viewing the aforementioned reports. That said, standard technology reports, such as browser, device, screen size or resolution, should work automatically.

Finally, because the `iframe` loads as a separate page and fully executes the JavaScript on that page, the AMP is not as lightweight as the AMP standard intended. To be clear, this does not affect page load time (the `iframe` loads after the page is done loading), but the CPU and network will be doing slightly more than they otherwise would, which might impact scrolling smoothness. In practice, we have not seen a large impact, but we are working with Google to minimize the user experience impact of this approach.

Summary

If you need click tracking and don't mind visitors being counted as entirely new visitors separate from your site, use the `"adobeanalytics"` tracking template, with our recommendation that you put the data into a *separate report suite*. If you need the Experience Cloud ID service, do not want visitor or visit inflation, and are okay with only firing Analytics on page load, we recommend using the `"adobeanalytics_nativeConfig"` solution.

Adobe Analytics is excited to partner with Google and our publishers to bring industry leading analytics capabilities to publishers on the mobile web in a blazing fast user experience. Although these two solutions currently offer their own tradeoffs, we are committed to building the best long term solution to answer the evolving analytics needs our customers have.

The AMP Project is moving fast and changes occur frequently, so check back [here](#) frequently for updates to our examples. What we've shown you here should be enough to get started, but expect changes as we improve our integrations further and as more publishers adopt AMP over time.

If you have questions or problems, please reach out to your Adobe Consultant or Customer Care.

Frequently Asked Questions

Question	Answer
Is video tracking available for either the <code>"adobeanalytics"</code> or <code>"adobeanalytics_nativeConfig"</code> template?	Unfortunately, not yet. The AMP standard supports only triggers for "visible", "click", and "timer", and does not yet support explicit triggers for video tracking that can be listened to by the <code>amp-analytics</code> tag. Also, because the <code>"adobeanalytics_nativeConfig"</code> tag can only be loaded once, it is not compatible with video viewing which occurs after the AMP has loaded.
You mention that visitor inflation is lower for the <code>"adobeanalytics_nativeConfig"</code> template in your comparison. What does that mean? What would cause visitor inflation in either the	<p>The <code>"adobeanalytics"</code> template does not allow Adobe Analytics to set a visitor identification cookie; this means all visits and visitors to your AMP page will be treated as a new and independent visit and visitor in your report suite.</p> <p>The <code>"adobeanalytics_nativeConfig"</code> template, however, allows the Adobe Analytics visitor identification cookie to be set in nearly all cases, except for</p>

Question	Answer
"adobeanalytics" or the "adobeanalytics_nativeConfig" solution?	new visitors using the Safari browser. This means that any visitors from Safari who have not previously visited a publisher's site will be inflated in Adobe Analytics reporting.
Should I use a separate report suite for AMPs?	We recommend using a separate report suite for AMPs if you use the adobeanalytics template, because of the visitor/visit inflation issue. However, we will also set the JavaScript version to "AMP vX.X" from the amp-analytics tag template so that you can segment that traffic out of a combined report suite if necessary.
What is the Experience Cloud ID service? Do I need it?	The Experience Cloud ID Service (formerly <i>visitor ID service</i>) enables Experience Cloud core services and allows integrations between different Adobe Experience Cloud solutions. If you have integrations with Adobe Audience Manager or Adobe Target, you are likely using this service. This service is also the foundation for many upcoming Adobe Analytics features. If you need ID service support or will need it in the future, we recommend using the <code>iframeMessage</code> solution.
For the "adobeanalytics_nativeConfig" template, where should I host my utility page?	The AMP standard does not allow for iframes to load from the exact domain and subdomain of the AMP itself. As such, we recommend that you host the utility page on a separate subdomain from your main site, especially if your company has its own CDN that plans on caching AMPs. For maximum compatibility, pick a subdomain such as <code>ampmetrics.publisher.com</code> that is apart from where the actual AMP content will reside.
Isn't this similar to Facebook Instant Articles? How do I setup Adobe Analytics with Facebook Instant Articles?	Facebook Instant Articles support a similar solution to the nativeConfig solution outlined above. In fact, the stats.html page created above can serve your analytics needs for both AMP and FIA simultaneously. For more information on implementing tracking on FIA, see Facebook Instant Articles

Facebook Instant Articles

How to implement Analytics in Facebook Instant Articles.

Facebook Instant Articles is a new method for publishers to build fast, interactive articles on Facebook. Instant Articles can load content up to 10 times faster than mobile web.

Adobe Analytics can be embedded within the Facebook Instant Articles to track visitor behavior as they interact with the content. Because publisher content is within the Facebook app and not on the publisher's websites, the tagging approach is slightly different than a standard Analytics implementation.

1. Embed an Analytics HTML page

When creating your Facebook Instant Article content, you can embed the analytics HTML content within an iFrame. For example:

```
<iframe class="no-margin" src="http://[your-domain-here]/analytics.html" height="0"></iframe>
```

2. Modify your Analytics HTML

The sample HTML below can be used to capture stats from the instant articles. This file would typically be hosted on one of your company's web servers. Each time an Instant Article is loaded, it loads the file in an iFrame as outlined in step one, which triggers sending the analytics data to Adobe.

```
<html>
  <head>
    <title>Facebook Stats</title>
    <script language="JavaScript" type="text/javascript"
src="http://[your-domain-here]/js/VisitorAPI.js"></script>
    <script language="JavaScript" type="text/javascript"
src="http://[your-domain-here]/js/AppMeasurement.js"></script>
  </head>
  <body>
    <script>
      //Standard Variable Configuration
      var v_orgId = "[your-marketing-cloud-org-id]@AdobeOrg";
      var s_account = "[your-report-suite-id]";
      var s_trackingServer = "[your-tracking-server]";
      var s_visitorNamespace = "[your-namespace]";

      //Instantiation
      var visitor = Visitor.getInstance(v_orgId);
      visitor.trackingServer = s_trackingServer;

      var s = s_gi(s_account);
      s.account = s_account;
      s.trackingServer = s_trackingServer;
      s.visitorNamespace = s_visitorNamespace;
      s.visitor = visitor;

      //Custom Variables
      s.pageName = s.Util.getQueryParam("pageName");
      s.prop10 = "Facebook Instant Article";

      //Page View Image Request Call
      s.t();
    </script>
  </body>
</html>
```

To modify this sample HTML for your specific implementation

1. It is recommended to host the latest copies of unmodified versions of the VisitorAPI.js and AppMeasurement.js on a common directory on your company's web servers.

These files can also be downloaded from within the Code Manager in the Analytics UI if required.

2. Include your Analytics implementation standard configuration variables:
 - a. Your Experience Cloud Org ID.
 - b. The report suite ID where the Facebook Instant Article traffic will be captured.
 - c. Your company's tracking server domain.
 - d. Your visitor namespace variable. **Note:** Many of these values can be found within your standard Analytics implementation. Customer Care or Adobe Consulting can assist in providing the proper values if necessary.
3. [Set custom variable and event tracking.](#)

4. Include the page view image request syntax (`s.t()`).

3. Set custom variable and event tracking

Custom variables and events can be tracked within your analytics HTML via different approaches. The first approach is to include JavaScript logic in your custom configuration, similar to what you might do with a standard Analytics implementation. For example, to set the value of a prop, simply use the same syntax that you would use in a normal implementation:

```
s.prop10 = "Facebook Instant Article";
```

You can also dynamically send variables to the iframe by leveraging query string parameters in the `iframe src` attribute. For example:

```
<iframe class="no-margin"
src="http://[your-domain]/analytics.html?prop1=dynamic%20article%20title&eVar1=facebook%20page%20name%20or%20page%20name%20respcid%20or%20campid%20or%20e"
height="0"></iframe>
```

Those query string parameters can subsequently be set in the custom variables section of your analytics HTML JavaScript by using the `Util.getQueryParam` function within the default `AppMeasurement` library, as follows:

```
s.pageName = s.Util.getQueryParam("pageName");
s.campaign = s.Util.getQueryParam("cmpId");
s.eVar1 = s.Util.getQueryParam("eVar1");
s.prop1 = s.Util.getQueryParam("prop1");
```

Visitor Tracking

As long as the Analytics HTML page is hosted on your web server, Adobe can support your existing privacy policy across all Facebook Instant Articles. This means that if an end user has opted out of tracking on your primary site, they will also be opted out of tracking on all your Facebook Instant Articles, with no additional steps required. Using this utility page also means that the Experience Cloud ID service (visitor ID) is supported so that you can integrate the metrics and variables captured on your Facebook Instant Articles with the rest of the Experience Cloud. (An example is for targeted advertising using Adobe Audience Manager).

Tracking Limitations

There are few issues that should be noted with this approach. Any DOM values that are typically only accessible via JavaScript on the Facebook Instant Article (such as referrer) will not be retrievable in the iframe for tracking. However, standard technology reports like browser, device, screen size or resolution should work normally. Moreover, the pageURL can be obtained by referencing `document.referrer` from your utility page.

What's Next?

Adobe Analytics is excited to partner with Facebook and our publishers to bring industry leading analytics capabilities to publishers on the mobile web in a blazing fast user experience. We are committed to building the best long term solution to answer the evolving analytics needs our customers have.

The Facebook Instant Article project is moving quickly and changes are happening all the time, so check back with us frequently for updates. Expect changes as we improve our integrations further and as more publishers adopt Facebook Instant Articles in the future.

If you have questions or problems, please reach out to your Adobe Consultant or Customer Care.

Additional Web and Mobile Measurement Libraries

Lists the available measurement libraries.

The following table outlines the libraries available to collect Analytics data across all available platforms. For more information, see [Data Collection in Analytics](#).

Location	Options
Web Browser	<p>All Experience Cloud customers have access to Dynamic Tag Management, which is the standard for deploying JavaScript and HTML page tags for all solutions to your website.</p> <p>Other ways of implementing JavaScript and HTML measurement are described in the Analytics Implementation Guide.</p>
Web Server	<p>You can use native PHP and Java libraries on your web server to send analytics data.</p> <p>The Data Insertion API lets you send XML data directly to the data collection server using HTTP POST and GET, and Data Sources lets you send delimited hit data directly to Analytics.</p> <ul style="list-style-type: none"> • PHP AppMeasurement • Java AppMeasurement • Data Insertion API • Data Sources
Mobile Device	<p>Native libraries are provided for iOS, Android, Windows Phone 8, Blackberry, Symbian, and others.</p> <ul style="list-style-type: none"> • iOS AppMeasurement 4.x (latest version) • iOS AppMeasurement 3.x • Android AppMeasurement 4.x (latest version) • Android AppMeasurement 3.x • Windows Phone 8 AppMeasurement 3.x (latest version) • Silverlight, .NET, XBOX, and Windows Phone 7 AppMeasurement
Flash	<p>Flash apps using ActionScript can be measured on the desktop and on the web.</p> <p>Flash, Flex, and OSMF AppMeasurement</p>
Desktop	<ul style="list-style-type: none"> • WinRT for Windows 8 AppMeasurement 3.x (latest version) • OS X AppMeasurement 3.x (latest version) • Java AppMeasurement • Silverlight, .NET, XBOX, and Windows Phone 7 AppMeasurement
Video	<p>Video measurement across all platforms is available in the following guides:</p> <ul style="list-style-type: none"> • Heartbeat video measurement (latest version) • Milestone video measurement

REST and SOAP Web Services (on [Developer Connection](#))

- [Getting Started](#)
- [API Documentation Home](#)

Implementing Analytics using HTML image tags

Implement Analytics using an HTML image tag (hardcoded image request).

The browser then requests the image. Data moves with this image request via variables in the query string of the image request. The JavaScript combines browser-level variables with page-level variables for a comprehensive data collection solution. In some cases, a fully server-created image tag is appropriate. The standard elements of a JavaScript-based implementation are listed as follows:

HTML Code	This portion consists of JavaScript code that is placed in HTML pages (or templates) that set the value of JavaScript variables.
JavaScript Library	This file contains common code that: <ul style="list-style-type: none"> • Queries the browser about various properties, such as JavaScript version, OS version, the size and resolution of monitor being used, and other variables • Encodes and concatenates all the variables into an image request () that transports these variables to the data collection servers. It then references a JavaScript library file which is loaded and executed.
<noscript> tag	A simplified version of the image request is placed within a <noscript> tag that executes if the user has disabled JavaScript, or does not have JavaScript capabilities. This part of the implementation is optional and generally applies to approximately 2% of the Internet population.

JavaScript can detect browser settings that are not available to a server, such as browser window height/width, monitor resolution, and Netscape plug-ins. By using a server-side method to create an image tag, these variables cannot be captured. The JavaScript sets a random number in the image request to overcome browser and proxy server caching. This allows all page views to be accurately tracked. In certain situations, server-side code has advantages over the JavaScript-based code, including the following:

- JavaScript is very accurate (98-100%). There are times when the utmost accuracy is desired, even in situations where a user quickly clicks to another page before the JavaScript has executed. Creating the image tag server-side increases the accuracy level by several percentage points.
- For tracking conversion events, such as purchases, where accuracy is very important.
- This strategy may also be used to fully populate the image request within the <noscript> tag for tracking users without JavaScript, or with JavaScript disabled.



Note: *The use of server-generated image tags requires additional time to implement, and is more difficult to debug, deploy, and maintain. Adobe strongly encourages clients to use JavaScript-based data collection on every page where possible. Various reports and features, including visitor click map, download links, exit links, and browser-based variables (browser width/height, etc.) cannot be collected or supported using this implementation method.*

Implementing without JavaScript Guidelines

There are additional requirements and configurations for implementing Analytics without JavaScript.

You can view sample code to further understand the implementation. The following table outlines the additional requirements and configurations:

Requirement	Description						
Case-Sensitive	The parameter names (<i>pageName</i> , <i>purchaseID</i> , and so forth) are case-sensitive and will not properly record data unless they appear as designated in the table displayed in Data Collection Query Parameters .						
Encode Query Parameters	<p>The values for each of the query string parameters must be URL encoded. URL encoding converts characters that are normally illegal when appearing in a query string, such as a space character, into an encoded character beginning with %. For example, a space character is converted into %20.</p> <p>The JavaScript version of this function is called <code>escape</code> (and to decode, <code>unescape</code>). Microsoft IIS Version 5.0 also includes an <code>Escape</code> and <code>Unescape</code> function for encoding query strings. Other Web server scripting languages also provide encoding/decoding utilities.</p>						
Maximum Variable Length	Each variable has a maximum length. This length is specified for each variable in Analytics Variables . Exceeding the maximum length for a variable causes the value of this variable to be truncated for storage and display in Analytics.						
Invalid Characters	Characters with character codes above decimal 128 are invalid, as are not-printing character codes under 128. HTML formatting (" <code><h1></code> ") is also invalid, as are trademark, registered trademark, and copyright symbols.						
Secure (https:> vs. Non-Secure (http:) Image Requests	<p>On pages that are accessed via https (secure protocol), the URL portion of the image request changes to accommodate a different set of data collection servers.</p> <p>The following table illustrates the different URLs used for secure and non-secure image requests.</p> <table border="1" data-bbox="618 1161 1471 1436"> <thead> <tr> <th data-bbox="618 1161 857 1213">Protocol</th> <th data-bbox="857 1161 1471 1213">URL</th> </tr> </thead> <tbody> <tr> <td data-bbox="618 1213 857 1346">https:</td> <td data-bbox="857 1213 1471 1346">https://namespace.<data center-specific...>.207.net/b/ss/ reportsuite /1/G.5--NS/...</td> </tr> <tr> <td data-bbox="618 1346 857 1436">http:</td> <td data-bbox="857 1346 1471 1436">http://namespace.<data center-specific URL*>.207.net/b/ss/ reportsuite /1/G.5--NS/...</td> </tr> </tbody> </table> <p>The * in the URL above denotes a data-center specific URL that is provided to you by your Adobe Consultant. Adobe uses several data centers, and it is necessary to implement the correct URL your organization has been assigned. Any code downloaded out of Admin Console within your company account has the correct data center supplied automatically. Code provided from external sources may need to be corrected in order to point to the correct data center.</p> <p>For clients who use multiple report suites, they should be listed only in the directory section, and not the domain section of the URL, as shown below.</p>	Protocol	URL	https:	https://namespace.<data center-specific...>.207.net/b/ss/ reportsuite /1/G.5--NS/...	http:	http://namespace.<data center-specific URL*>.207.net/b/ss/ reportsuite /1/G.5--NS/...
Protocol	URL						
https:	https://namespace.<data center-specific...>.207.net/b/ss/ reportsuite /1/G.5--NS/...						
http:	http://namespace.<data center-specific URL*>.207.net/b/ss/ reportsuite /1/G.5--NS/...						

Requirement	Description						
	<table border="1" data-bbox="618 201 1471 443"> <thead> <tr> <th data-bbox="618 201 870 258">Protocol</th> <th data-bbox="870 201 1471 258">URL</th> </tr> </thead> <tbody> <tr> <td data-bbox="618 258 870 352">https:</td> <td data-bbox="870 258 1471 352">https://102.112.207.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...</td> </tr> <tr> <td data-bbox="618 352 870 443">http:</td> <td data-bbox="870 352 1471 443">http:// <u>suite1</u> .112.207.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...</td> </tr> </tbody> </table> <p data-bbox="618 474 1471 611">The * in the URL above denotes a data-center specific URL that is provided to you by your Adobe Consultant. Adobe uses several data centers, and it is necessary to implement the correct URL to which your organization has been assigned.</p>	Protocol	URL	https:	https://102.112.207.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...	http:	http:// <u>suite1</u> .112.207.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...
Protocol	URL						
https:	https://102.112.207.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...						
http:	http:// <u>suite1</u> .112.207.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...						
URL and Referring URL	<p data-bbox="618 646 1471 831">The URL and Referring URL may be populated from the server in the g= and r= variables. Use the Request ServerVariables (HTTP_REFERER) or Request ServerVariables (URL) (IIS/ASP), or the appropriate variable for your server/scripting technology. The referring URL (r=) is extremely important for tracking referring URLs, domains, search engines, and search terms.</p> <p data-bbox="618 852 1471 989">If <i>pageName</i> is not being used, it is imperative that the Current URL field is uniquely populated. If neither <i>pageName</i> nor Current URL (g=) is populated, the record is invalid and is not processed. At a minimum, the URL is a required field in order to process the record.</p>						
Effects of Caching	<p data-bbox="618 1045 1471 1182">HTML and other Web pages can be cached by browsers or servers that are between the visitor and the website that is serving the content. Caching prevents an accurate count of page views and other events unless a "cache-busting" technique is employed.</p> <p data-bbox="618 1203 1471 1308">Adobe's standard JavaScript includes a dynamic method of changing the image request to avoid page and image caching, allowing an accurate count of page views.</p> <p data-bbox="618 1329 1471 1476">However, in creating a server-side image request, this randomization does not occur. Page reloads and cached pages (either in the browser's cache or in a proxy server) are not counted in certain cases when using server-side image requests.</p> <p data-bbox="618 1497 1471 1644">SSL (https:) pages are not, by definition, ever cached so this warning applies only to non-secure (http:) pages. Additionally, pages with parameters (<code>http://www.samplesite.com/page.asp?parameter=1</code>) or certain file extensions (<code>.asp</code>, <code>.jsp</code>, etc.) are also not cached.</p> <p data-bbox="618 1665 1471 1843">The examples below also illustrate a minimal JavaScript solution that primarily assembles the image request server-side, and tacks on a random number in the browser. This method overcomes the caching that would otherwise be encountered on static HTML pages accessed via the http: protocol.</p>						

Requirement	Description
nameSpace Variable	<p>The nameSpace query string parameter is required for non-JavaScript implementations.</p> <p>Example: ns=nameSpace</p> <p>Contact your Adobe Consultant or Account Manager to obtain your organization's nameSpace value.</p>

Sample Code

Examples to illustrate the use of a server-generated image tag within a HTML sample page.

The table below displays the values used in the sample.

Variable	Value
pageName	Order Confirmation
Current URL	https://www.somesite.com/cart/confirmation.asp
events	purchase,event1
c1	Registered
purchaseID	0123456
products	Books;Book Name;1;19.95
state	CA
zip	90210
a random #	123456

Example 1

The example below displays a server-side image tag. The highlighted random number prevents caching of the image.

```
<html>
<head>
</head>
<body>
Order Confirmation<br>
Thanks for your order #0123456.

</body>
</html>
```

Example 2

The example below shows a minimal JavaScript image tag.

```
<html>
<head>
</head>
<body>
Order Confirmation<br>
Thanks for your order #0123456.
```

```
<script language="javascript"><!--
s.s_date = new Date();
s.s_rdm = s.s_date.getTime();
s.s_desturl="<img width=\"1\" height=\"1\"
src=\"https://102.112.207.net/b/ss/suite1,suite2/1/G.4--NS/\" + s.s_rdm +
\"?pageName=Order%20Confirmation&events=purchase%20Event1&c1=Registered
&purchaseID=0123456&products=Books%3BBook%20Name%3B1%3B19.95&state=CA&zip=90210&g=http
s%3A//www.somesite.com/cart/confirmation.asp\">";
document.write(s.s_desturl);
//--></script>
</body>
</html>
```

Mobile Tracking over WAP and I-Mode Protocols

This section describes the tracking methodology for legacy mobile devices that do not use http browsers.

Mobile Network Protocols

WAP and I-Mode are the two major protocols or standards used today. WAP is mostly used in the US, and I-Mode is popular in Japan and Europe.

Sites must often be designed separately for different protocols. The Adobe image tag does not need to be customized for each protocol.

WAP 1.0	WAP 1.0 is quickly losing popularity in the US. This required pages be built in WML, and had its own protocol.
WAP 2.0	Most phones are WAP 2.0 compliant, meaning they support XHTML MP (a mobile version of XHTML).
I-Mode	I-Mode supports CHTML (compact HTML) and does not download third-party images. First-party collection domain implementations should always be used with I-Mode sites so images are downloaded.

Mobile Protocol Network Gateway

When a mobile device requests a page from a Web server, the request is sent through a gateway, which converts the mobile request (usually in the WAP or I-Mode protocol) into an HTTP request that is sent to a Web server.

When a mobile device requests a page from a Web server, the request is sent through a gateway, which converts the mobile request (usually in the WAP or I-Mode protocol) into an HTTP request that is sent to a Web server. The Web server responds to the gateway, which forwards the request to the phone. This gateway acts much like a standard proxy server. The gateway does, however, pass the user agent string of the mobile device on to the Web server. This lets the Web server respond with a page that is built specifically for the device requesting it.

Because screen size on WAP mobile devices is so limited, mobile pages are very light, often containing only text and one cached image. When the image tag is added to the pages, a request is sent on every page for an image from Adobe servers. When the image, a WBMP, is returned, Adobe servers instruct the browser not to cache it. Consequently, the image is requested again on subsequent pages. The random number described above should be used to protect against browsers that do not obey the Adobe no-cache directives.

Tagging Pages For Mobile Protocols

Mobile tracking code is placed on the page in the form of a server-generated image tag.

Mobile tracking code is placed on the page in the form of a server-generated image tag. Because scripting languages like JavaScript and WMLScript are not generally supported across mobile devices, the tracking beacon cannot be generated dynamically via a scripting language.

• While the mobile beacon image is actually 2x2 pixels, to ensure support for all mobile devices, you should set height and width properties to 5. For example:

```

```

Use a Random Number to Prevent Caching

While Adobe servers instruct browsers not to cache the tracking beacon, not all browsers are guaranteed to follow those instructions. To further prevent caching of the beacon, it is recommended that the Web server dynamically generate a random number in the image URL path. Doing so insures greater accuracy of reporting. The random number should be in the last section of the path, as shown here:

```
.
```

Include an ALT Tag

Some mobile browsers require that all images have alt text included in the image tag. The following shows how the ALT attribute should appear in the image tag:

```
.
```

It is important that the /5/ always appears correctly in the path. This is used by Adobe servers to identify mobile devices. If the standard /1/ is used, Adobe servers attempt to set a cookie on the mobile device.

Change the Default Image Type

If the default image type is not supported on a particular device, no data is returned. To avoid this, you can force the Adobe data collection server to return a particular graphic type that the mobile device supports. The code following the report suite name specifies the image type:

- /5/ returns the default image type.
- /5.1/ or /1/ always returns a GIF image.
- /5.5/ always returns a WBMP image.

See [Identifying Mobile Devices](#).

Reports for Devices Using Mobile Protocols

Because mobile devices are tracked via a beacon, just like other visitors, most reports are available and correct.

VISTA can be used to alter data collected from both Mobile and standard methods. All **CustomInsight (prop and eVar)**, **Event**, **Site Traffic**, and **Pathing** reports are supported.

Search Engines, Search Keywords, Referring Domains, and Referrers

These reports only have data if the referrer is populated in the image request sent from the mobile page. The referrer is populated via the "r" query string parameter, as outlined in the Implementing without JavaScript white paper. You must also manually pass the referrer information into the image request.

The 'r' query string parameter must include the protocol of the referrer. If the protocol is left off, the referrer report is not populated. For example, use `r=http://msn.com` not `r=msn.com`.

Geosegmentation and Domains

Geosegmentation reports are based on the IP address of the device sending the request. Because mobile devices rely on a gateway to request images from Adobe servers, the gateway's IP address is used to determine the geo-location of the user. Because gateways and their IP addresses are registered for large networks, the associated geo-location is often less accurate.

Domains are also based on the IP address of the gateway, which means the domains report often contains the name of the carrier who owns the gateway. Due to Mobile Virtual Network Operators (MVNOs), this may not be accurate.

Connection Types

Adobe maintains a known range of IP addresses that belong to mobile carriers. When a hit is received from an IP range that belongs to a known mobile carrier, the hit appears as "Mobile Carrier" on the Connection Type Report. Otherwise, mobile traffic is listed under "Lan/Wifi".

Time Zones, Cookies, Java, JavaScript, Monitor Colors and Resolutions, Browser Width and Height, and Netscape Plug-ins

These reports are all collected by using JavaScript to detect specific settings of the browser. Because JavaScript is not used to create the image beacon on mobile devices, data collected from mobile users is not included in these reports.

Custom Link Measurement on Mobile Protocols

Many mobile device users download files to their devices such as podcasts, ring tones, and similar files. Because many mobile devices do not support JavaScript, link measurement must be implemented through redirects. To use redirects, you must modify the href links in the html to include the REDIR element. The general format for a custom link is as follows:

```
http://<your_Namespace>.112.207.net/b/ss/<RSID>/4/REDIR/  
?url=<destination_URL>&pe=<link_type>&pev1=<current_URL>&pev2=<link_name>
```

Keep in mind the following when creating link reference:

- The URL value must be URL encoded.
- You should set a `pageName` or `page URL (g)` parameter.
- Dynamic variables can read the URL parameter but not set it.
- If no cookie is set, Adobe servers perform a normal cookie handshake.
- To track links, you must include `pe`, `pev1`, and `pev2` parameters.

A custom link measurement URL looks similar to the following:

```
<a href=" http://johnny_appleseed.122.207.net/b/ss/appleseedpodcasts/4/REDIR/  
?url=http%3A%2F%2Fwww.johnny_appleseed.org%2Fmpegs%2Fplanting_apple_trees.mpeg&pe=lnk_d  
&pev1=http%3A%2F%2Fwww.johnny_appleseed.org%2Fmpegs%2Fplanting_apple_trees.mpeg&pev2=pl  
anting_apple_trees&">Planting an Apple Tree</a>
```

For more information, see the [Exit Link Tracking Redirects whitepaper](#).

Variables for Analytics Implementation and Reporting

Analytics provides a number of variables to collect Analytics data. For example, the value in the `pageName` variable is the name of the Web page being reported. This section lists the variables that are supported by AppMeasurement.

For a listing of how each variable appears in Analytics reports, see [Variables - How They Are Used in Reporting](#).

How to Set Variables

AppMeasurement requires that all configuration variables be set before the initial call to the track function, *t()*. If configuration variables are set after the call to *t()*, unexpected results may occur.

Configuration variables are set inside the *doPlugins* function, which is called during the execution of the track function. The specific configuration variable causing this issue is *trackInlineStats*, which enables ClickMap data collection. This leaves the ClickMap module in an indeterminate state, which results in the first tracking call appending the string "undefined" to the Adobe Analytics beacon, affecting the currency code.

To resolve this issue, move all configuration variables above the *doPlugins* function.

```

/***** CONFIG SECTION *****/
/* Ensure these variables are correct before deploying */
var s_account="[INSERT-REPORT-SUITE-ID-HERE]"
var s=s_gi(s_account)
s.trackingServer="[INSERT-TRACKING-SERVER-HERE]"
s.visitorNamespace="[INSERT-VISITOR-NAMESPACE-HERE]"
s.charset="ISO-8859-1"
s.currencyCode="USD"
s.trackDownloadLinks=true
s.trackExternalLinks=true
s.trackInlineStats=true
s.linkDownloadFileTypes="exe,zip,wav,mp3,mov,mpg,avi,wmv,doc,pdf,xls"
s.linkInternalFilters="javascript:,three,two,one,dev16,.,nike"
s.linkLeaveQueryString=false
s.linkTrackVars="None"
s.linkTrackEvents="None"
s.debugTracking=true

s.usePlugins=true;
function s_doPlugins(s) {
    //add your custom plugin code here
}
s.doPlugins=s_doPlugins;

/*
===== DO NOT ALTER ANYTHING BELOW THIS LINE ! =====

AppMeasurement for JavaScript version: 1.5.3
Copyright 1996-2016 Adobe, Inc. All Rights Reserved
More info available at http://www.omniture.com
*/
function AppMeasurement(){var a=this;a.version="1.5.3";var
k=window;k.s_c_in||(k.s_c_il=[],k.s_c_in=0);a._il=k.s_c_il;a._in=k.s_c_in;a._il[a._in]=a;k.s_c_in++;a._c="s_c";var
q=k.AppMeasurement.zb;q||(q=null);var
r=encodeURIComponent("r="+encodeURIComponent(a._il.join("&")));var
0>a.indexOf(b)?a:a.split(b).join(d)};a.escape=function(c){var b,d;if(!c)return
c=encodeURIComponent(c);for(b=0;b<b+1;b++)d="+-"+!*(c).substring(b,b+1),0<c.indexOf(d)&&(c=a.replace(c,d,"%"+d.charCodeAt(0).toString(16).toUpperCase()));return
c};a.unescape=function(c){if(!c)return c;c=0<c.indexOf("+")?a.replace(c,"+"," "):c;try{return
decodeURIComponent(c)}catch(b){}return unescape(c)};a.gb=function(){var
c=k.location.hostname,b=a.fpCookieDomainPeriods,d;b||(b=a.cookieDomainPeriods);if(c&&!a.cookieDomain&&

```

Configuration Variables

Configuration variables set in the *AppMeasurement.js*.

Configuration variables control the way data is captured and processed in reporting. The most-common configuration variables that are typically set in the main global JavaScript (*AppMeasurement.js*). These variables can be set within the Reports & Analytics page-level code and links when appropriate.

Not all of these variables appear in the code by default when you generate code through the **Admin Tool > Code Manager**. Some of these configuration variables may not be applicable to your site's implementation needs.

Some of the goals of using these configuration variables are:

- Track multiple sites/domains.
- Use any currency on purchases.
- Capture data indifferent languages.
- Link tracking (number of downloaded files, links to external sites).
- Track custom links for unique purposes.



Note: *AppMeasurement* requires that all configuration variables are set before the initial call to the track function, *t()*. If configuration variables are set after the call to *t()*, unexpected results may occur. To ensure proper data collection, all configuration variables must be above the *doPlugins* function.

s.account

The *s.account* variable determines the report suite where data is stored and reported.

If sending to multiple report suites (multi-suite tagging), *s.account* may be a comma-separated list of values. The report suite ID is determined by Adobe.

Parameters

Max Size	Debugger Parameter	Reports Populated	Default Value
40 Bytes	In the URL path	N/A	N/A

Each report suite ID must match the value created in the Admin Console. Each report suite ID must be 40 bytes or less, but the aggregate of all report suites (the entire comma-separated list) has no limit.

The report suite is the most fundamental level of segmentation in reporting. You can set as many report suites as your contract allows. Each report suite refers to a dedicated set of tables that are populated in Adobe's collection servers. A report suite is identified by the *s_account* variable in your JavaScript code.

Within Analytics, the site drop-down box in the upper left of the reports displays the current report suite. Each report suite has a unique identifier called a report suite ID. The *s_account* variable contains one or more report suite IDs to which data is sent. The report suite ID value, which is invisible to Analytics users, must be provided or approved by Adobe before you use it. Every report suite ID has an associated "friendly name" that can be changed in the report suites section of the Admin Console.

The *s_account* variable is normally declared inside the JavaScript file (*s_code.js*). You can declare the *s_account* variable on the HTML page, which is a common practice when the value of *s_account* may change from page to page. Because the *s_account* variable has a global scope, it should be declared immediately before including Adobe's JavaScript file. If *s_account* does not have a value when the JavaScript file is loaded, no data is sent to Analytics.

Adobe's DigitalPulse Debugger displays the value of *s_account* in the path of the URL that appears just below the word "Image," just after */b/ss/*. In some cases, the value of *s_account* also appears in the domain, before *112.2o7.net*. The value in the path is the only value that determines the destination report suite. The bold text below shows the report suites that data is sent to, as it appears in the debugger. See .

[http://mycompany.112.207.net/b/ss/mycompanycom,mycompanysection/1/H.1-pdv-2/s21553246810948?\[AQB\]](http://mycompany.112.207.net/b/ss/mycompanycom,mycompanysection/1/H.1-pdv-2/s21553246810948?[AQB])

Syntax and Possible Values

The report suite ID is an alphanumeric string of ASCII characters, no more than 40 bytes in length. The only non-alphanumeric character allowed is a hyphen. Spaces, periods, commas and other punctuation are not allowed. The `s_account` variable may contain multiple report suites, all of which receive data from that page.

```
var s_account="reportsuitecom[,reportsuite2[,reportsuite3]]"
```

All values of `s_account` must be provided or approved by Adobe.

Examples

```
var s_account="mycompanycom"
```

```
var s_account="mycompanycom,mycompanysection"
```

Configuring the Variable in Analytics

The friendly name associated with each report suite ID can be changed by Adobe Customer Care. The friendly name can be seen in Analytics in the site drop-down box in the top, left section of the screen.

Pitfalls, Questions, and Tips

- If `s_account` is empty, not declared, or contains an unexpected value, no data is collected.
- When the `s_account` variable is a comma-separated list (multi-suite tagging), do not put spaces between report suite IDs.
- If **s.dynamicAccountSelection** is set to 'True,' the URL is used to determine the destination report suite. Use the DigitalPulse Debugger to determine the destination report suite(s).
- In some cases, VISTA can be used to alter the destination report suite. Using VISTA to re-route or copy the data to another report suite is recommended when using first-party cookies, or if your site has more than 20 active report suites.
- Always declare `s_account` inside the JS file or just before it is included.

s.dynamicAccountSelection

The `dynamicAccountSelection` variable lets you dynamically select the report suite based on the URL of each page.



Note: `dynamicAccountSelection` does not work with custom link tracking.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	False



Note: Both `dynamicAccountList` and `dynamicAccountMatch` are ignored if the `dynamicAccountSelection` variable is not declared or set to 'false.'

Syntax and Possible Values

```
s.dynamicAccountSelection=[true|false]
```

Only 'true' and 'false' are allowed as values of `dynamicAccountSelection`.

Examples

```
s.dynamicAccountSelection=true
```

```
s.dynamicAccountSelection=false
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Dynamic account selection is not supported by [About AppMeasurement for JavaScript](#).
- Always use the DigitalPulse Debugger to determine which report suite is receiving data from each page.

s.dynamicAccountList

AppMeasurement for JavaScript can dynamically select a report suite to which it sends data. The *dynamicAccountList* variable contains the rules used to determine the destination report suite.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	""

This variable is used in conjunction with the *dynamicAccountSelection* and *dynamicAccountMatch* variables. The rules in *dynamicAccountList* are applied if *dynamicAccountSelection* is set to 'true,' and they apply to the section of the URL specified in *dynamicAccountMatch*.

If none of the rules in *dynamicAccountList* matches the URL of the page, the report suite identified in *s_account* is used. The rules listed in this variable are applied in a left-to-right order. If the page URL matches more than one rule, the left-most rule is used to determine the report suite. As a result, your more generic rules should be moved to the right of the list.

In the following examples, the page URL is `http://www.mycompany.com/path1/?prod_id=12345`, *dynamicAccountSelection* is set to 'true,' and *s_account* is set to "mysuitecom."

DynamicAccountList Value	DynamicAccountMatch Value	Report Suite to Receive Data
<code>ms2=www2.mycompany.com;ms1=mycompany.com</code>	<code>window.location.host</code>	mysuite1
<code>"mysuite1=path4,path1;mysuite2=path2"</code>	<code>window.location.pathname</code>	mysuite1, mysuite2
<code>"mysuite1=path5"</code>	<code>window.location.pathname</code>	mysuitecom, mysuite1
<code>"myprodsuite=prod_id"</code>	<code>window.location.search?window.location.search"?)</code>	myprodsuite

Syntax and Possible Values

The *dynamicAccountList* variable is a semicolon-separated list of name=value pairs (rules). Each piece of the list should contain the following items:

- one or more report suite ID (separated by commas)
- an equals sign
- one or more URL filters (comma-separated)

```
s.dynamicAccountList=rs1[,rs2]=domain1.com[,domain2.com/path][;...]
```

Only standard ASCII characters should be used in the string (no spaces).

Examples

```
s.dynamicAccountList="mysuite2=www2.mycompany.com;mysuite1=mycompany.com"
```

```
s.dynamicAccountList="ms1,ms2=site1.com;ms1,ms3=site3.com"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Dynamic account selection is not supported by [About AppMeasurement for JavaScript](#).
- If the page URL matches multiple rules, the furthest rule on the left is used.
- If no rules match, the default report suite is used.
- If your page is saved to someone's hard drive or translated via a web-based translation engine (such as Google's translated pages), the dynamic account selection probably won't work. For more precise tracking, populate the `s_account` variable server-side.
- The `dynamicAccountSelection` rules apply only to the section of the URL specified in `dynamicAccountMatch`.
- When using dynamic account selection, be sure to update `dynamicAccountList` every time you obtain a new domain.
- Use the DigitalPulse Debugger when trying to identify the destination report suite. The `dynamicAccountSelection` variable always overrides the value of `s_account`.

s.dynamicAccountMatch

The `dynamicAccountMatch` variable uses the DOM object to retrieve the section of the URL to which all rules in `dynamicAccountList` are applied.

This variable is only valid when `dynamicAccountSelection` is set to 'True.' Since the default value is `window.location.host`, this variable is not required for **Dynamic Account Selection** to work. For additional information, see [dynamicAccountList](#).

The rules found in `dynamicAccountList` are applied to the value of `dynamicAccountMatch`. If `dynamicAccountMatch` only contains `window.location.host` (default), the rules in `dynamicAccountList` apply only to the domain of the page.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	<code>window.location.host</code>

Syntax and Possible Values

The `dynamicAccountMatch` variable is usually populated by the Adobe consultant who provides the AppMeasurement for JavaScript file. However, the values listed below may be applied at any time.

```
s.dynamicAccountMatch=[DOM object]
```

Description	Value
Domain (default)	<code>window.location.host</code>
Path	<code>window.location.pathname</code>
Query String	<code>(window.location.search?window.location.search:"?")</code>
Domain and Path	<code>window.location.host+window.location.pathname</code>
Path and Query String	<code>window.location.pathname+(window.location.search?window.location.search:"?")</code>
Full URL	<code>window.location.href</code>

Examples

```
s.dynamicAccountMatch=window.location.pathname
```

```
s.dynamicAccountMatch=window.location.host+window.location.pathname
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Dynamic account selection is not supported by [About AppMeasurement for JavaScript](#).
- When pages are saved to a hard drive, `window.location.host` is empty, causing those page views to be sent to the default report suite (in `s_account`).
- When a page is translated via a web-based translation engine, such as Google, the **Dynamic Account Selection** does not work as designed. For more precise tracking, populate the `s_account` variable server-side.

s.dynamicVariablePrefix

The `dynamicVariablePrefix` variable allows deployment to flag variables, which should be populated dynamically.

Cookies, request headers, and image query string parameters are available to be populated dynamically.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	D=	Any	D=

Syntax and Possible Values

```
s.prop1="D=User-Agent"
```

OR USE CUSTOM FLAG FOR DYNAMIC VARIABLES

```
s.dynamicVariablePrefix=".."
```

Examples

```
s.prop1="D=User-Agent"
```

OR USE CUSTOM FLAG FOR DYNAMIC VARIABLES

```
s.dynamicVariablePrefix=".."
```

```
s.prop1="..User-Agent"
```

Pitfalls, Questions, and Tips

- Dynamic variables can be used to significantly reduce the total length of the URL by copying values into other variables.
- Dynamic variables can be used to collect data from headers and cookies not otherwise available for data collection.

s.charSet

The `charSet` variable translates the character set of the Web page into UTF-8.

The `charSet` property, which is normally set in the JavaScript file, is used by Analytics to convert incoming data into UTF-8 for storage and reporting by Analytics.



Note: The `charSet` property is required when sending data to a multi-byte report suite and should never be used with a standard report suite. Setting the `charSet` property with a standard ISO report suite can result in variable truncation or unexpected character conversion.

The value of the `charSet` property should match the web page encoding in the META tag or http header, even though the syntax may differ slightly. Although the META tag may use an alias for the encoding, the value of `charSet` should use the preferred (or official) name of the encoding.

Some of the more common encodings with their preferred name and aliases are listed in the following table.

Preferred Name	Aliases
ISO-8859-1	ISO-8859-1, latin1
ISO-8859-2	ISO-8859-2, latin2
ISO-8859-5	ISO-8859-5, cyrillic
Big5	Big-5
Shift_JIS	SJIS

Because numerous encodings and aliases exist, contact your Implementation Consultant or Adobe Customer Care to confirm the proper value for `charSet` if it does not appear in the table above.

If a site has different web encodings on different pages, or a single JavaScript file is used for multiple sites, the `charSet` property can be set to a default value in the JavaScript file and then reset on specific pages as needed to override the default; for example, `s.charSet="UTF-8"` or `s.charSet="SJIS"`.

Any non-blank value of the `charSet` parameter will cause data to be converted into UTF-8 for storage. Any characters in the 128-255 range will be converted to the proper UTF-8 two-byte sequence and stored. These characters will not display properly in a standard report suite. Therefore, the `charSet` property should never be used with a standard report suite.

Likewise, a blank value of the `charSet` parameter will bypass the data conversion process, and any characters in the range 128-255 will be stored as a single byte. These characters will not display properly in a multi-byte report suite since the single-byte codes for these characters are not valid UTF-8. Therefore, the `charSet` parameter should always be used with a multi-byte report suite. Additionally, the proper value should be used with respect to the web page encoding.

If the `charSet` variable contains an incorrect value, the data in all other variables are translated incorrectly. If JavaScript variables on your pages (e.g. `pageName`, `prop1`, or `channel`) contain only ASCII characters, `charSet` does not need to be defined. However, if the variables on your pages contain non-ASCII characters, the `charSet` variable must be populated.

Parameters

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	CE	N/A	" "

Syntax and Possible Values

```
s.charSet="character_set"
```

Examples

```
s.charSet="ISO-8859-1"
```

```
s.charSet="SJIS"
```

s.currencyCode

The `currencyCode` variable determines the conversion rate to be applied to revenue.

All monetary amounts are stored in a currency of your choice. If that currency is the same as that specified in *currencyCode*, or *currencyCode* is empty, no conversion is applied.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	cc	Conversion > Purchases > Revenue All Conversion reports showing Revenue or monetary values	"USD"

If your site allows visitors to purchase in multiple currencies, you should use the *currencyCode* variable to make sure revenue is stored in the appropriate currency. For example, if the base currency for your report suite is USD, and you sell an item for 40 Euros, you should populate the *currencyCode* with "EUR" on the HTML page. As soon as data collection receives the data, it uses the current conversion rate to convert the 40 Euros to its USD equivalent.

Populating the *currencyCode* variable on the HTML page instead of in the JavaScript file is recommend if you sell in multiple currencies. If you want to use your own conversion rates rather than the conversion rates used by Adobe, set the *currencyCode* to equal the base currency of your report suite. You then convert all revenue before sending it into Analytics.

Currency conversion applies to both revenue and any currency events. These are events that are used to sum values similar to revenue, such as tax and shipping. The revenue and currency events are specified in the products string. For more information on products, see [events](#). For more details on how currencies are managed, see [Multi-Currency Support](#).

Syntax and Possible Values

```
s.currencyCode="currency_code"
```

Only the currency codes listed in [Multi-Currency Support](#) are allowed.

Examples

```
s.currencyCode="GBP"
```

```
s.currencyCode="EUR"
```

Configuration Settings

Adobe Customer Care can change the default currency setting for your report suite. When you change the base currency for a report suite, the existing revenue in the system is not converted. All new revenue values will be converted accordingly.

Pitfalls, Questions, and Tips

- If you notice surprisingly large amounts of revenue in reports, ensure that the *currencyCode* variable and base currency of the report suite are set correctly.
- The *currencyCode* variable is not persistent, meaning that the variable must be passed in the same image request as any revenue or other currency-related metrics.
- Currency events should not be used for non-currency purposes. If you need to count arbitrary or dynamic values that are not currency, use the **numeric** event type.
- When the *currencyCode* variable is empty, no conversion is applied.

s.cookieDomain

The `cookieDomain` variable determines the domain on which the Analytics cookies `s_cc` and `s_sq` are set.

Commonly, `s.cookieDomainPeriods` is used to generate `s.cookieDomain` from `window.location.hostname`. Instead of using `s.cookieDomainPeriods`, you can explicitly set `s.cookieDomain` to what you want to use in your implementation. For example, you could set cookies at the fully qualified page-name using:

```
s.cookieDomain = window.location.hostname;
```

s.cookieDomainPeriods

The `cookieDomainPeriods` variable determines the domain on which the Analytics cookies `s_cc` and `s_sq` are set by determining the number of periods in the domain of the page URL. This variable is also used by some plug-ins in determining the correct domain to set the plug-in's cookie.

The default value for `cookieDomainPeriods` is "2". This is the value that is used if `cookieDomainPeriods` is omitted. For example, using the domain `www.mysite.com`, `cookieDomainPeriods` should be "2". For `www.mysite.co.jp`, `cookieDomainPeriods` should be "3".

If `cookieDomainPeriods` is set to "2" but the domain contains three periods, the JavaScript file attempts to set cookies on the domain suffix.

For example, if setting `cookieDomainPeriods` to "2" on the domain `www.mysite.co.jp`, the `s_cc` and `s_sq` cookies are created on the domain `co.jp`. Because `co.jp` is an invalid domain, almost all browsers reject these cookies. As a consequence, visitor click map data is lost, and the **Visitor Profile > Technology > Cookies** report indicates that almost 100% of visitors reject cookies.

If `cookieDomainPeriods` is set to "3" but the domain contains only two periods, the JavaScript file sets the cookies on the subdomain of the site. For example, if setting `cookieDomainPeriods` to "3" on the domain `www2.mysite.com`, the `s_cc` and `s_sq` cookies are created on the domain `www2.mysite.com`. When a visitor goes to another subdomain of your site (such as `www4.mysite.com`), all cookies set with `www2.mysite.com` cannot be read.



Note: Do not include additional subdomains as part of `cookieDomainPeriods`. For example, `store.toys.mysite.com` would still have `cookieDomainPeriods` set to "2". This variable definition correctly sets the cookies on the root domain, `mysite.com`. Setting `cookieDomainPeriods` to "3" in this example would set cookies on the domain `toys.mysite.com`, which has the same implications as the prior example.

See also [s.fpCookieDomainPeriods](#).

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	CDP	Affects multiple reports as it controls how the visitor ID is stored and handled.	"2"



Note: Some cloud computing services are considered *Top-Level Domains*, which do not allow cookies to be written. (For example, `*.compute.amazonaws.com`, `*.herokuapp.com`, `*.googlecode.com`, and so on.) If you implement on those services, you could potentially be affected by Analytics privacy setting that removes users who have blocked all cookies if you don't have your own domain set up (for example, if you're testing your implementation). In this case, any hit where the system has determined that cookies are disabled, non-functional, or inaccessible is opted out and thus excluded from reporting.

Examples

Setting the variable manually:

```
s.cookieDomainPeriods = "3";
```

Several examples to dynamically set the variable if your core JavaScript file hosts both types:

```
document.URL.indexOf(".co.") > 0 ? s.cookieDomainPeriods = "3" : s.cookieDomainPeriods = "2";
```

```
s.cookieDomainPeriods = "2";
var d=window.location.hostname;
if(d.indexOf(".co.uk") > 0 || d.indexOf(".com.au") > 0)
  {s.cookieDomainPeriods = "3";}
```

```
s.cookieDomainPeriods = "2";
if(window.location.indexOf(".co.jp") > 0 || window.location.indexOf(".com.au") > 0)
  {s.cookieDomainPeriods = "3";}
```

Pitfalls, Questions, and Tips

- If you notice that visitor click map data is absent, or that the **Traffic > Technology > Cookies** report shows a large percentage of visitors who reject cookies, check that the value of *cookieDomainPeriods* is correct.
- If *cookieDomainPeriods* is higher than the number of sections in the domain, cookies will be set with the full domain. This can cause data loss as visitors switch between subdomains.
- The *cookieDomainPeriods* variable was used in deprecated implementations prior to *trackingServer* to set the visitor ID cookie. Though only present in outdated code, failure to correctly define *cookieDomainPeriods* in this circumstance puts your implementation at risk of data loss.

s.fpCookieDomainPeriods

The *fpCookieDomainPeriods* variable is for cookies set by JavaScript (*s_sq*, *s_cc*, plug-ins) that are inherently first-party cookies even if your implementation uses the third-party 2o7.net or omtrdc.net domains.

The *fpCookieDomainPeriods* variable should never be dynamically set. If you use *cookieDomainPeriods*, it is good practice to specify a value for *fpCookieDomainPeriods* as well. *fpCookieDomainPeriods* inherits the *cookieDomainPeriods* value. Note that *fpCookieDomainPeriods* does not affect the domain on which the visitor ID cookie is set, even if your implementation treats this as a first-party cookie.

The name "*fpCookieDomainPeriods*" refers to the number of periods (".") in the domain when the domain begins with "www." For example, *www.mysite.com* contains two periods, while *www.mysite.co.jp* contains three periods. Another way to describe the variable is the number of sections in the main domain of the site (two for *mysite.com* and three for *mysite.co.jp*).

The AppMeasurement for JavaScript file uses the *fpCookieDomainPeriods* variable to determine the domain with which to set first-party cookies other than the **visitor ID** (*s_vi*) cookie. There are at least two cookies affected by this variable, including *s_sq* and *s_cc* (used for visitor click map and cookie checking respectively). Cookies used by plug-ins such as **getValOnce** are also affected.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	cookieDomainPeriods

Sample Code for Setting Cookie Domain Variables

```
s.fpCookieDomainPeriods="2"
var d=window.location.hostname
if(d.indexOf('.co.uk')>-1||d.indexOf('.com.au')>-1)
  s.fpCookieDomainPeriods="3"
```

Syntax and Possible Values

The *cookieDomainPeriods* variable is expected to be a string, as shown below.

```
s.fpCookieDomainPeriods="3"
```

Examples

```
s.fpCookieDomainPeriods="3"
```

```
s.fpCookieDomainPeriods="2"
```

Configuration Settings

None

s.cookieLifetime

The *cookieLifetime* variable is used by both JavaScript and data collection servers in determining the lifespan of a cookie.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	cl	Traffic > Technology > Cookies All visitor-related reports	""

If *cookieLifetime* is set, it overrides any other cookie expirations for both JavaScript and data collection servers, with one exception, described below. The *cookieLifetime* variable can have one of three values:

- Analytics Cookies
- Cookies
- JavaScript Settings and Plugins

Syntax and Possible Values

```
s.cookieLifetime="value"
```

The possible values are listed as follows:

- ""
- "NONE"
- "SESSION"
- An integer representing the number of seconds until expiration

Examples

```
s.cookieLifetime="SESSION"
```

```
s.cookieLifetime="86400" // one day in seconds
```

Configuration Settings

None

Pitfalls, Questions, and Tips

cookieLifetime affects Analytics tracking. If, for example, *cookieLifetime* is two days, then monthly, quarterly, and yearly unique visitor reports will be incorrect. Use caution when setting *cookieLifetime*.

s.doPlugins

The *doPlugins* variable is a reference to the *s_doPlugins* function, and allows the *s_doPlugins* function to be called at the appropriate location within the JavaScript file.

The *s_doPlugins* function is called each time any of the following occurs:

- The *t()* function is called
- The *tl()* function is called
- An exit or download link is clicked
- Any page element being tracked by visitor click map is clicked

The *doPlugins* function is used to run customized routines to gather or alter data. If you are using an object name other than "s," make sure that the *s_doPlugins* is renamed appropriately. For example, if your object name is *s_mc*, the *s_doPlugins* function should be called *s_mc_doPlugins*.

Syntax and Possible Values

The *s_doPlugins* function should not be in quotes, and *doPlugins* should always be assigned to the exact name of the *s_doPlugins* function (if that function is renamed).

```
s.doPlugins=s_doPlugins;
```

Examples

```
s.doPlugins=s_doPlugins;
```

```
s_mc.doPlugins=s_mc_doPlugins;
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- The only reason to change the object name (such as from *s* to *s_mc*) is if you share content with or pull content from other customers. Renaming the *s_doPlugins* function to **s_mc_doPlugins** ensures that another client's JavaScript file does not overwrite your *doPlugins* function.
- If you unexpectedly start pulling in content from another Adobe customer, and your *s_doPlugins* function is being overwritten, it is possible to simply rename the *s_doPlugins* function without changing the object name. While the best solution is to use a different object name than other JavaScript files on the same page, doing so is not required.

s.trackDownloadLinks

Set *trackDownloadLinks* to 'true' if you would like to track links to downloadable files on your site.

If *trackDownloadLinks* is 'true,' *linkDownloadFileTypes* is used to determine which links are downloadable files.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	True

The *trackDownloadLinks* variable should only be set to 'false' if there are no links to downloadable files on your site, or you don't care to track the number of clicks on downloadable files. If *trackDownloadLinks* is 'true,' when a file download link is clicked, data is immediately sent to Analytics. The data that is sent with a download link includes the link download URL, and visitor click map data for that link. If *trackDownloadLinks* is 'false,' then visitor click map data for links to downloadable files on your site is likely to be under reported.

Syntax and Possible Values

The *trackDownloadLinks* variable is expected to be either 'true' or 'false.'

Examples

```
s.trackDownloadLinks=true
```

```
s.trackDownloadLinks=false
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- When *trackDownloadLinks* is 'false,' links that people use to download files on your site are likely to be under reported in visitor click map.
- When *trackDownloadLinks* is 'true,' data is sent each time a visitor clicks a file download link.

s.trackExternalLinks

If *trackExternalLinks* is 'true,' *linkInternalFilters* and *linkExternalFilters* are used to determine whether any link clicked is an exit link.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	True

The *trackExternalLinks* variable should only be set to 'false' if there are no exit links on your site, or if you don't care to track the number of clicks on those exit links. An exit link is any link that takes a visitor off of your site. If *trackExternalLinks* is 'true,' then when you click an exit link, tracking data is immediately sent. The data that is sent with an exit link includes the link URL, link name, and visitor click map data for that link. If *trackExternalLinks* is 'false,' then visitor click map data for exit links on your site is likely to be under reported.

Syntax and Possible Values

The *trackExternalLinks* variable is expected to be either 'true' or 'false.'

```
s.trackExternalLinks=true|false
```

Examples

```
s.trackExternalLinks=true
```

```
s.trackExternalLinks=false
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- When *trackExternalLinks* is 'false,' links that take people away from your site are likely to be under reported in visitor click map.
- When *trackExternalLinks* is 'true,' data is sent each time a visitor clicks on an exit link (before link target loads).

s.trackInlineStats

The *trackInlineStats* variable determines whether ClickMap data is gathered.

If *trackInlineStats* is 'true,' data about the page and link clicked are stored in a cookie called s_sq. If 'false,' s_sq will have a value of "[[B]]," which is considered null.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	ClickMap	False

Syntax and Possible Values

```
s.trackInlineStats=true|false
```

The *trackInlineStats* variable is expected to be either 'true' or 'false.'

Examples

```
s.trackInlineStats=true
```

```
s.trackInlineStats=false
```

Configuration Settings

None

s.linkDownloadFileTypes

The *linkDownloadFileTypes* variable is a comma-separated list of file extensions.

If your site contains links to files with any of these extensions, the URLs of these links will appear in the **File Downloads** report.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	Traffic > Site Traffic > File Downloads	".exe,zip,wav,mp3,mov,mpg,avi,wmv,doc,pdf,xls"

The *linkDownloadFileTypes* variable is only relevant when *trackDownloadLinks* is set to 'True.'

Only left-mouse-clicks on a link are counted in the **File Downloads** report. All file downloads that start automatically when a page loads, or that are only downloaded after a redirect, are not counted in the **File Downloads** report.

When you right-click a file and select the "Save Target As..." option, it is not counted in the **File Downloads** report.

The *linkDownloadFileTypes* variable may be used to track clicks to RSS feeds. If you have links to RSS feeds with a .xml or other extension, appending ".xml" to the *linkDownloadFileTypes* list allows you to see how often each RSS link is clicked.

The following sections contain more information:

- [Syntax and Possible Values](#)
- [Examples](#)
- [Configuration Settings](#)
- [Pitfalls, Questions, and Tips](#)

Syntax and Possible Values

Only include file extensions (no spaces).

```
s.linkDownloadFileTypes="type1[,type2[,type3[...]]]"
```

Any file extension may be included in the list. Be careful not to include a common file extension, such as htm or aspx, in *linkDownloadFileTypes*. Doing so causes an extra image request to be sent for each click, which will be billed as a primary server call.

Examples

```
s.linkDownloadFileTypes="exe,zip,wav,mp3,mov,mpg,avi,wmv,doc,pdf,xls"
```

```
s.linkDownloadFileTypes="exe,zip,wav,mp3,mov,mpg,avi,wmv,doc,pdf,xls,xml"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Only left-clicks on download files cause the URL to appear in the **File Downloads** report.
- Including a common file extension in *linkDownloadFileTypes* may significantly increase the total server calls sent to Adobe's servers.
- Links to server-side redirects or HTML pages that automatically begin downloading a file are not counted unless the file extension is in *linkDownloadFileTypes*.
- Links that use JavaScript (such as `javascript:openLink()`) are not counted in file downloads.

s.linkInternalFilters

The *linkInternalFilters* variable is used to determine which links on your site are exit links.

It is a comma-separated list of filters that represent the links that are part of the site.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	Paths > Entries & Exits > Exit Links	



Note: We had previously suggested setting the *linkInternalFilters* to `javascript:`. However, this resulted in all domains being considered external, including the current domain on which the tag resides. If you want several domains to be considered internal, you can add those, as shown in the examples below.

The *linkInternalFilters* variable is used to determine whether a link is an exit link, which is defined as any link that takes a visitor away from your site. Whether the target window of an exit link is a pop-up, or the existing window, does not affect whether the link appears in the exit links report. Exit links are only tracked if *trackExternalLinks* is set to `true`. (See [Link Tracking](#) in the Dynamic Tag management documentation for information about how DTM handles exit links.) The filters in *linkInternalFilters* are not case-sensitive.

The list of filters in *linkInternalFilters* applies to the domain and path of any link by default. If *linkLeaveQueryString* is set to `true`, then the filters apply to the entire URL (domain, path, and query string). The filters are always applied to the absolute path of the URL, even if a relative path is used as the href value.

Be careful that all the domains of your site (and any partners who are using your JavaScript file) are included in *linkInternalFilters*. If you do not have all domains included in the list, all links on and to those domains are considered exit links, increasing the server calls sent. If you would like multiple domains or companies to use a single

AppMeasurement for JavaScript file, you may consider populating *linkInternalFilters* on the page, overriding the value specified in the JavaScript file. If you have vanity domains that immediately redirect to your main domain, those vanity domains do not need to be included in the list.

The following example illustrates how this variable is used. In this example, the URL of the page is

```
http://www.mysite.com/index.html.
```

```
s.trackExternalLinks=true
s.linkInternalFilters="mysite.com"
s.linkExternalFilters=""
s.linkLeaveQueryString=false
...
<a href="http://www.mysite.com">Not an Exit Link</a>
<a href="/careers/job_list.html">Not an Exit Link</a>
<a href="http://www2.site3.com">Exit Link</a>
<a href="http://www2.site1.com/partners/">Exit Link</a>
```

Syntax and Possible Values

The *linkInternalFilters* variable is a comma-separated list of ASCII characters. No spaces are allowed.

```
s.linkInternalFilters="site1.com[,site2.com[,site3.net[...]]]"
```

Examples

```
s.linkInternalFilters="mysite.com"
```

```
s.linkInternalFilters="mysite.com,mysite.net,vanity1.com"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Include all domains that the AppMeasurement for JavaScript file may be served under in the filter list.
- Periodically check the **Paths > Entries & Exits > Exit Links** report to make sure that none of the entries in that report are incorrect.
- Periodically review partner contracts to determine if they contain restrictions on link tracking. For example, you might be prohibited from tracking links that appear in partner display ads. Filter partner links by adding their domain to *linkInternalFilters*:

```
s.linkInternalFilters="mysite.com,mysite.net,mypartner.net/adclick"
```

s.linkLeaveQueryString

By default, query strings are excluded from all reports.

For some exit links and download links, the important portion of the URL can be in the query string, as shown in the following sample URL.

```
http://www.mycompany.com/download.asp?filename=myfile.exe
```

The download file name can be defined in the query string and, consequently, the query string is necessary to make the **File Downloads** report more accurate.

The *linkLeaveQueryString* variable determines whether or not the query string should be included in the **Exit Links** and **File Download** reports.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	Exit Links File Downloads	false



Note: Setting `linkLeaveQueryString=true` includes all query string parameters for all exit links and download links.

Syntax

```
s.linkLeaveQueryString=[false/true]
```

Examples

```
s.linkLeaveQueryString=false
```

Possible Values

```
s.linkLeaveQueryString=false
```

```
s.linkLeaveQueryString=true
```

Configuring the Variable

No configuration is necessary for this variable.

Pitfalls, Questions, and Tips

- Setting `s.linkLeaveQueryString=true` includes all query string parameters for all exit links and download links.
- The `linkLeaveQueryString` variable does not affect recorded page URLs, visitor click map, or **Path** reports.

s.linkTrackVars

The `linkTrackVars` variable is a comma-separated list of variables that are sent with custom, exit, and download links.

If `linkTrackVars` is set to "", all variables that have values are sent with link data. To avoid inflation of instances or page views associated with other variables, Adobe recommends populating `linkTrackVars` and `linkTrackEvents` in the **onClick** event of a link that is used for link tracking.

All variables that should be sent with link data (custom, exit, and download links) should be listed in `linkTrackVars`. If `linkTrackEvents` is used, `linkTrackVars` should contain "events."

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	Any	"None"

When populating `linkTrackVars`, do not use the 's.' prefix for variables. For example, instead of populating `linkTrackVars` with "s.prop1," you should populate it with "prop1." The following example illustrates how `linkTrackVars` should be used.

```
s.linkTrackVars="eVar1,events"
s.linkTrackEvents="event1"
s.events="event1"
s.eVar1="value A"
s.eVar2="value B"
s.t() // eVar1, event1 and event2 are recorded
<a href="http://google.com">event1 and eVar1 are recorded</a>
```

```
<a href="test.php" onClick="s=s_gi('rs1');s.eVar1='value C';s.events='';s.tl(this,'o')">eVar1 is recorded</a>
```

Because the link to `google.com` is an exit link (unless you are Google), `event1` and `eVar1` are sent with the exit link data, increasing the instances associated with `eVar1` and the number of times `event1` is fired. In the link to `test.php`, **eVar1** is sent with a value of 'value C' because that is the current value of **eVar1** at the time that `tl()` is called.

Syntax and Possible Values

The `linkTrackVars` variable is a case-sensitive, comma-separated list of variable names, without the object name prefix. Use 'eVar1' instead of 's.eVar1.'

```
s.linkTrackVars="variable_name[,variable_name[...]]"
```

The `linkTrackVars` variable may contain only variables that are sent to Analytics, namely: `events`, `campaign`, `purchaseID`, `products`, **eVar1-75**, **prop1-75**, **hier1-5**, `channel`, `server`, `state`, `zip`, and `pageType`.

Examples

```
s.linkTrackVars="events,prop1,eVar49"
```

```
s.linkTrackVars="products"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- If `linkTrackVars` is blank, all variables that have values are tracked with all server calls.
- Any variable listed in `linkTrackVars` that has a value at the time of any download, exit, or custom link, are tracked.
- If `linkTrackEvents` is used, `linkTrackVars` must contain "events."
- Do not use the "s." or "s_objectname." prefix for variables.

s.linkTrackEvents

The `linkTrackEvents` variable is a comma-separated list of events that are sent with a **custom**, **exit**, or **download** link.

If an event is not in `linkTrackEvents`, it is not sent to Analytics, even if it is populated in the **onClick** event of a link, as shown in the following example:

```
s.linkTrackVars="events"
s.events="event1,event2"
s.t() // both event1 and event2 are recorded
<a href="help.php" onClick="s=s_gi('rs1');s.tl(this,'o')">event1 is recorded</a>
<a href="test.php" onClick="s=s_gi('rs1');s.events='event2';s.tl(this,'o')">No events are recorded</a>
```

In the first link to `help.php`, notice that the `events` variable retains the value that was set before the link was clicked. This allows `event1` to be sent with the custom link. In the second example, the link to `test.php`, `event2` is not recorded because it is not listed in `linkTrackEvents`.

To avoid confusion and potential problems, Adobe recommends populating `linkTrackVars` and `linkTrackEvents` in the **onClick** event of a link that is used for link tracking.

The `linkTrackEvents` variable contains the events that should be sent with **custom**, **download**, and **exit** links. This variable is only considered if `linkTrackVars` contains "events."

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	Conversion	"None"

Syntax and Possible Values

The *linkTrackEvents* variable is a comma-separated list of events (no spaces).

```
s.linkTrackEvents="event1[,event2[,event3[...]]]"
```

Only event names are allowed in *linkTrackEvents*. These events are listed in [Events](#). If a space appears before or after the event name, the event can not be sent with any link image requests.

Examples

```
s.linkTrackEvents="purchase,event1"
```

```
s.linkTrackEvents="scAdd,scCheckout,purchase,event14"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- The JavaScript file only uses *linkTrackEvents* if *linkTrackVars* contains the "events" variable. "events" should be included in *linkTrackVars* only when *linkTrackEvents* is defined.
- Beware if an event is fired on a page, and is listed in *linkTrackEvents*. That event is recorded again with any **exit**, **download**, or **custom** links unless the events variable is reset prior to that event (in the **onClick** of a link or after the call to the *t()* function).
- If *linkTrackEvents* contains spaces between event names, the events are not recorded.

s.linkExternalFilters

If your site contains many links to external sites, and you do not want to track all exit links, use *linkExternalFilters* to report on a specific subset of exit links.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	Paths > Entries & Exits > Exit Links	""

The *linkExternalFilters* variable is an optional variable used in conjunction with *linkInternalFilters* to determine whether a link is an exit link. An exit link is defined as any link that takes a visitor away from your site. Whether the target window of an exit link is a popup or the existing window, it does not affect whether the link appears in the exit links report. Exit links are tracked only if *trackExternalLinks* is set to 'true.' The filters in *linkExternalFilters* and *linkInternalFilters* are case insensitive.



Note: If you don't want to use *linkExternalFilters*, delete it or set it to "".

The filters list in *linkExternalFilters* and *linkInternalFilters* apply to the domain and path of any link by default. If *linkLeaveQueryString* is set to 'true,' the filters apply to the entire URL (domain, path, and query string). These filters are always applied to the absolute path of the URL, even if a relative path is used as the href value.

Most companies find that *linkInternalFilters* gives them enough control over exit links that they don't need *linkExternalFilters*. Using *linkExternalFilters* simply decreases the likelihood that an exit link is considered external.

If *linkExternalFilters* has a value, then a link is considered only external if it does not match *linkInternalFilters* and does match *linkExternalFilters*.

The following example illustrates how this variable is used. In this example, the URL of the page is `http://www.mysite.com/index.html`.

```
s.trackExternalLinks=true
s.linkInternalFilters="javascript:,mysite.com"
s.linkExternalFilters="site1.com,site2.com,site3.com/partners"
s.linkLeaveQueryString=false
...
<a href="http://www.mysite.com">Not an Exit Link</a>
<a href="/careers/job_list.html">Not an Exit Link</a>
<a href="http://www2.site3.com">Not an Exit Link</a>
<a href="http://www.site1.com">Exit Link</a>
<a href="http://www2.site3.com/partners/offer.asp">Exit Link</a>
```

Syntax and Possible Values

The *linkExternalFilters* variable is a comma-separated list of ASCII characters. No spaces are allowed.

```
s.linkExternalFilters="site1.com[,site2.com[,site3.net[...]]]"
```

Any portion of a URL might be included in *linkExternalFilters*, separated by commas.

Examples

```
s.linkExternalFilters="partnersite.com,partnertwo.net/path/"
```

```
s.linkExternalFilters=""
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Using *linkExternalFilters* can result in fewer links on your site being exit links. Do not use this variable in place of *linkInternalFilters* to force internal links to become exit links.
- If *linkExternalFilters* should be applied to the query string of a link, make sure *linkLeaveQueryString* is set to 'true.' See [linkLeaveQueryString](#) before setting to "true".
- To disable exit link tracking, set *trackExternalLinks* to "false".

s.usePlugins

If the *s_doPlugins* function is available and contains useful code, **s_usePlugins** should be set to 'true.'

When **usePlugins** is 'true,' the *s_doPlugins* function is called prior to each image request.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	True

Syntax and Possible Values

```
s.usePlugins=true|false
```

The **usePlugins** variable is expected to be either 'true' or 'false.'

Examples

```
s.usePlugins=true
```

```
s.usePlugins=false
```

The **usePlugins** variable should only be false (or not declared) if the `s_doPlugins` function is not declared in your JavaScript file.

Configuration Settings

None

Context Data Variables

Context data variables let you define custom variables on each page that can be read by processing rules.

Instead of explicitly assigning values to props and eVars in your code, you can send data in context data variables that are mapped using processing rules. Processing rules provide a powerful graphical interface to make changes to data as it is received. Based on the values sent in context data, you can set events, copy values to eVars and props, and execute additional conditional statements.



Note: Context data variables are not case sensitive. For example, the following 2 variables are effectively identical:

```
s.contextData['article_title'] = 'Weekend Concert Controversy';
```

and

```
s.contextData['ARTICLE_TITLE'] = 'Weekend Concert Controversy';
```

Using context data helps prevent you from making code updates to support different report suite configurations.

For example, you can define the following `s.contextData` variable:

```
s.contextData['myco.rsid'] = 'value'
```

Using processing rules you can add a condition that checks for a `myco.rsid` context data variable. When this variable is found, you can add an action to copy it to a prop or eVar.

Context data variables can also be defined directly in the processing rules interface to temporarily store a value, or to collect values from a context data variable you know will be used on the report suite. For example, if you need to swap two values, you could create a context data variable to store a value during the swap.

Since processing rules are applied only when data is collected, it is important to set up processing rules before you start sending context data. Context data values that are not read by processing rules when a hit is processed are discarded.

Rules

Rule	Description
Supported names and characters	Context data variable names can contain only alphanumeric characters, underscores and dots. Any additional characters are stripped out. Context data variables do not have a numeric designation. Rather, they are named.

Rule	Description
	For example, the context data variable <code>login_page-home</code> automatically becomes <code>login_pagehome</code> . All data sent to the <code>login_page-home</code> variable is allocated under <code>login_pagehome</code> .
Namespace	A good practice is to prefix your variables with your company name, site name, or a similar value to make sure the name is unique across your report suite. Context data variables can be named similar to other JavaScript variables. Be aware that the namespace <code>a.*</code> is reserved for use by Adobe products in context variable names. For example, the AppMeasurement Library for iOS uses <code>a.InstallEvent</code> to measure application installations.
URL Limits for Internet Explorer	You might encounter a older URL limitation for Internet Explorer 6 and 7, where URLs are truncated at 2000 bytes. You can use the DigitalPulse debugger to determine the size of a URL string. With the recent updates to AppMeasurement (September 2014), HTTP POST is used with Internet Explorer 8+, which eliminates truncation issues.
Supported AppMeasurement version	Context data variables require at least H23 code or higher.

Sending Context Data on a Track Link Call

Include `ContextData` + the name of the variable that you would like included in `s.linkTrackVars`:

```
s.contextData['myco.value'] = "some value";
s.linkTrackVars = "contextData.myco.value";
s.tl(true, "o", "Link Name");
```

Examples

Possible ways to replace implementation of the `s.pageName` variable, assuming that processing rules are set up correctly for each:

```
s.contextData['page'] = "Home Page"
s.contextData['pagename'] = document.title // Takes the web page's title and passes it into
the pageName context data variable
s.contextData['pagevar'] = s.pageName // This example would be considered redundant, as both
the pageName and contextData variable are available in Processing rules
```

Other examples to implement context data variables:

```
s.contextData['owner'] = "Jesse"
s.contextData['campaign'] = "Campaign A"
s.contextData['author'] = "Sheridan Andrius"
```

For an example, see [Copy a Context Data Variable to an eVar](#) in Analytics Reference.

Dynamic Variables

Dynamic variables let you copy values from one variable to another without typing the full values multiple times in the image requests on your site.

Dynamic variables are used when capturing the same data (for example, campaign tracking codes) in multiple variables concurrently. This is a common practice in cases where different reports offer unique and important metrics. For example, capturing internal search keywords in a **Custom Conversion** variable and in a **Custom Traffic** variable lets you view the **Revenue** and the **Weekly Unique Visitors** metrics associated with these keywords, respectively.

Dynamic variables are also useful for viewing data under various reporting conditions. A campaign tracking code can be captured in multiple eVars with various allocation and cookie expiration settings. This lets users choose the way they want to attribute conversion metrics to these campaigns.



Note: Dynamic variables are not supported in conjunction with cookies (`s_cc`, `s_sq`, `s_fid`, `s_vi` and any cookie that is set by a plugin). You can not use `D=<cookie value>`.

A significant benefit of dynamic variables is the ability to capture long strings of data in multiple variables without actually passing the long string repeatedly. Some browsers limit the maximum length of HTTP GET requests (including the Adobe image request). Using dynamic variables ensures that all data is captured by reducing the length of the request to Adobe servers in cases where data is duplicated across several variables.

In the Adobe image request that occurs on the page view, if you are using dynamic variables to copy the value of **Custom Traffic 1** to **Custom Conversion 1**, you would see `v1=D=c1`. If eVar1 received a value previously in the request, Adobe's servers dynamically copy the value of **Custom Traffic 1** to **Custom Conversion 1** during data processing. As a result, the value originally passed using **Custom Traffic 1** also appears in the **Custom Conversion 1** reports.

Dynamic variables are passed by setting a variable to the desired value and then setting other variables to `D=[variable abbreviation]`. For abbreviations for each variable, see [Data Collection Query Parameters](#). Dynamic variables can pull data from the following locations:

- Other query-string variables
- HTTP headers (except for the Cookie HTTP header)

To create a dynamic variable, add a special prefix to the start of the value. The default prefix is "D=". There are two methods of flagging dynamic variables:

- Use a default prefix of D= (For example: `s.prop1="D=User-Agent"`)
- For non-JavaScript implementations, you can define a custom prefix using the "D" query-string parameter. For example, if the query-string parameter is `&D=$`, you can create a dynamic variable with the following command:
`s.prop1="$User-Agent"` .

The variable abbreviation used must match the variable parameter name passed in the image request. Some variables have multiple accepted parameters used in different cases. For example, `pageName=` and `gn=` both pass the page name, but the latter is most often used in mobile and hard-coded implementations. If the image request uses `pageName=` to pass the page name, then `D=pageName` is acceptable but `D=gn` is not. If the image request uses `gn=`, then `D=gn` is acceptable, but `D=pageName` is not.

The following information applies to dynamic variables:

- Dynamic variables work with all versions of AppMeasurement code.
- Dynamic variables are case sensitive.
- Dynamic variables support literal strings contained in quotes.
- Dynamic variable replacement occurs before processing rules, VISTA, and other processing.
- The dynamic variable prefix "D=" must be at the start of the variable value not in the middle. For example, use `c2='D="test7"+User-Agent'` rather than `c2=' "test7"+D=User-Agent'` .

- As with all implementation techniques, Adobe strongly recommends testing dynamic variable implementations heavily in a development environment before deploying to production. Because the full strings that are copied are not visible in client-side debugging tools, review the affected Analytics reports to confirm successful implementation.
- When copying values between variables with different maximum lengths, note that copying a value that exceeds the maximum length of the destination variable causes truncation. For example, **Custom Traffic** variables have 100-character limits and **Custom Conversion** variables have 255-character limits. When copying a 150-character value from `s.eVar1` to `s.prop1` using dynamic variables, this value is truncated in the **Custom Traffic** report at 100 characters.

Dynamic Variable Examples


Examples of dynamic variables you can use in Analytics.

In the Adobe image request that occurs on the page view, if you are using dynamic variables to copy the value of **Custom Traffic 1** to **Custom Conversion 1**, you would see `v1=D=c1`. If `eVar1` received a value previously in the request, Adobe's servers dynamically copy the value of **Custom Traffic 1** to **Custom Conversion 1** during data processing. As a result, the value originally passed using **Custom Traffic 1** also appears in the **Custom Conversion 1** reports.

Note that the `D=[variable]` value should be in quotes. The Analytics code treats this as a string. The string will be URL encoded when passed into Analytics (as you will see if viewing the request in the DigitalPulse Debugger or a similar utility). This is normal. Adobe's servers recognize the `D=[variable]` construction and will copy the appropriate value when they encounter this string.



Note: When using the image request to track links, the type of link (`download=lnk_d`, `exit=lnk_e`, or `custom link=lnk_o`) must be defined, as does the Link URL/Name (`pev2`). Links require manual implementation by inserting code within the `<a href>` tag.

 **Note:** Dynamic variables are not supported in conjunction with cookies (`s_cc`, `s_sq`, `s_fid`, `s_vi` and any cookie that is set by a plugin). You can not use `D=<cookie value>`.

JavaScript Example	Description
<code>s.eVar1="D=pageName"</code>	Captures the <code>pageName</code> value in <code>eVar1</code> .
<code>s.prop1='D=v2+": "+c2'</code>	Concatenates <code>eVar2:prop2</code> into <code>prop1</code> .
<code>s.prop1=s.eVar1="D=g"</code>	Passes the page URL into both <code>prop1</code> and <code>eVar1</code> .
<code>s.eVar1="D=v0"</code>	Captures the campaign in <code>eVar 1</code> .
<code>s.prop1='D=User-Agent+" - "+Accept-Language'</code>	Concatenates the user agent and accept language headers in <code>prop1</code> .
<code>s.prop1="D=User-Agent"</code>	Captures the user agent in <code>prop1</code> ,

Image Request Example	Description
<code>/b/ss/rsid/?gn=Home&D=~~&c1=~~v0</code> <code>/b/ss/rsid/?gn=Home&D=~~&c1=~~campaign</code> <code>/b/ss/rsid/?gn=Home&c1=D%3dv0%3d is</code> <code>/b/ss/rsid/?gn=Home&c1=%5b%5bv0%5d%5d%5b</code>	Four ways to set <code>prop1</code> to a campaign
<code>/b/ss/rsid/?gn=Home&D=~~&c2=~~x-up-subno</code>	Pulls the <code>x-up-subno</code> header into <code>prop2</code>
<code>&c1=D%3DUser-Agent</code>	Makes <code>prop1</code> a dynamic variable filled in with the HTTP header <code>User-Agent</code>
<code>&c1=D%3D%22test%22</code>	Makes <code>prop1</code> a dynamic variable filled in with the string "test". This becomes more useful when used with concatenation which utilizes the <code>+</code> operator.
<code>&c1=D%3D%22US%3A%20%22%2BUser-Agent</code>	Makes <code>prop1</code> a dynamic variable filled in with the <code>User-Agent</code> prefixed by "UA:"
<code>&vid=D%3DX-TM-ANTID</code>	This example searches for a unique header, which in this case is <code>X-TM-ANTID</code> .

Page Variables

Page variables directly populate a report, such as `pageName`, `List Props`, `List Variables`, and so on.

Additional eVars and Events

If you want to track additional information, but don't have enough variables to do so, you now have access to additional eVars and success events:



Note: JavaScript H-Code does not support these additional eVars and events.

Entitlements to Custom Dimensions and Events

Adobe Analytics Product			
Adobe Analytics - Foundation	75 props	200 eVars	1000 Events
Adobe Analytics - <i>Select</i>	75 props	200 eVars	1000 Events
Adobe Analytics - <i>Prime</i>	75 props	200 eVars	1000 Events
Adobe Analytics - <i>Ultimate</i>	75 props	250 eVars	1000 Events

Populating Variables and Events

- Variables can be populated in the data collection library if you are using these versions of AppMeasurement:
 - AppMeasurement for JavaScript version 1.4+
 - AppMeasurement for Flash version 3.9+
 - AppMeasurement for Java version 1.2.8+
 - AppMeasurement for Silverlight/.NET version 1.4.2+
 - AppMeasurement for PHP version 1.2.2+
- If you are on an earlier version of code, or on JavaScript H.*, you can populate context data and use processing rules to populate variables.
- All versions of data collection code can populate events 101-1000.
- Processing rules can be used to populate eVars 76-250 based on context data or other variables.

Frequently Asked Questions

• Will all Adobe Analytics interfaces have immediate access to these new variables?

These interfaces will have immediate access: Analysis Workspace, Reports & Analytics, Report Builder, Ad Hoc Analysis, APIs, and Data Workbench.

• Will these additional eVars and events automatically show up in my data feeds?

Data feeds will have access to the new variables and events once enabled. New eVar columns will not appear until you choose to include them. However, new events will appear in the event_list column as soon as they're enabled, and the events lookup table contains the event names for those event IDs. Do not enable new events unless you are ready to consume them in Data Feeds.

• How do I request new data feed columns?

To request new columns, refer to [Configuring Data Feeds](#).


• What if I am an Analytics Ultimate customer who wants to go back to Analytics Prime and I currently have more than 200 eVars enabled?

Adobe will not disable any of your existing eVars, but you will not be able to enable more. If you disable eVars, you cannot turn them back on until you are below the Analytics Prime limit of 200 enabled eVars.

browserHeight

The *browserHeight* variable displays the height of the browser window.

This variable is populated after the page code and before *doPlugins* is run.

 **Note:** *This variable should only be read and never set.*

You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.


Parameters

Query Param	Value	Example	Reports Affected
bh	A positive integer	865	Traffic > Technology > Browser Height

browserWidth

The *browserWidth* variable displays the width of the browser window.

This variable is populated after the page code and before *doPlugins* is run.

 **Note:** *This variable should only be read and never set.*

You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Parameters

Query Param	Value	Example	Reports Affected
bw	A positive integer	1179	Traffic > Technology > Browser Width

campaign

The *campaign* variable identifies marketing campaigns used to bring visitors to your site. The value of *campaign* is usually taken from a query string parameter.

Parameters

Max Size	Debugger Parameter	Reports Populated	Default Value
255 Bytes	v0	Conversion > Campaigns > Tracking Code	""

Every element in a marketing campaign should have an associated unique tracking code. For example, a paid search engine keyword may have a tracking code of 112233. When someone clicks the keyword with the 112233 tracking code and is routed to the corresponding website, the *campaign* variable records the tracking code.

There are two main ways to populate the *campaign* variable:

- The **getQueryParam** plug-in, used in the JavaScript file, retrieves a query string parameter from the URL. For more information on the **getQueryParam** plug-in, see [Implementation Plug-ins](#).
- Assign a value to the *campaign* variable in the HTML on the Web page.

With either method of populating the *campaign* variable, the Back button traffic may inflate the actual number of click-throughs from a campaign element.

For example, a visitor enters your site by clicking a paid search keyword. When the visitor arrives on the landing page, the URL contains a query string parameter identifying the tracking code for the keyword. The visitor then clicks a link to another page, but then immediately clicks the Back button to return to the landing page. When the visitor arrives a second time on the landing page, the URL with the query string parameter identifies the tracking code again. And a second click-through is registered, thereby falsely inflating the number of click-throughs.

To avoid this inflation of click-throughs, Adobe recommends using the **getValOnce** plug-in to force each campaign click-through to be counted only once per session. For more information on the **getValOnce** plug-in, see [Implementation Plug-ins](#).

Syntax and Possible Values

```
s.campaign="112233"
```

The *campaign* variable has the same limitations as all other variables. Adobe recommends limiting the value to standard ASCII characters.

Case Sensitivity

eVars are case insensitive, but they are displayed in the capitalization of the first occurrence. For example, if the first instance of eVar1 is set to "Logged In," but all subsequent instances are passed as "logged in," reports always show "Logged In" as the value of the eVar.

Examples

```
s.campaign="112233"
```

```
s.campaign=s.getQueryParam('cid');
```

Configuration Settings

Each campaign value remains active for a user, and receives credit for that user's activities and success events until it expires. You can change the expiration of the campaign variable in the Admin Console.

Pitfalls, Questions, and Tips

- To keep click-throughs from being inflated, use the **getValOnce** plug-in to let the click-through for a campaign be counted only once per session. For more information on the **getValOnce** plug-in, see [Implementation Plug-ins](#).
- For more information on tracking marketing campaigns and keyword buys, see [Campaigns](#).
- Use the DigitalPulse Debugger to see the actual value of campaigns (v0 in the debugger). If v0 does not appear in the debugger, no campaign data is recorded for that page.

channel

The *channel* variable is most often used to identify a section of your site.

For example, a merchant may have sections such as Electronics, Toys, or Apparel. A media site may have sections such as News, Sports, or Business.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 Bytes	CH	Site Content > Site Sections	""

Adobe recommends populating the channel variable on every page. You can also turn on a correlation between the *channel* and **page name** variables.

When sections have one or more levels of subsections, you can show those sections in the *channel* variable or use separate variables to identify levels.

Syntax and Possible Values

```
s.channel="value"
```

The *channel* variable has no extra limitations on its values.

Examples

```
s.channel="Electronics"
```

```
s.channel="Media"
```


Pitfalls, Questions, and Tips

If your site contains multiple levels, you can use the *hierarchy* or another variable to designate those levels. The *channel* value does not persist, but the success events fired on the same page are attributed to the *channel* value.

colorDepth

The *colorDepth* variable is used to show the number of bits used to display color on each pixel of the screen.

For example, 32 represents 32 bits of color on the screen. This variable is populated after the page code and before *doPlugins* is run.

 **Note:** This variable should only be read and never set.


You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example	Reports Affected
c	8,16, and 32	32	Traffic > Technology > Monitor Color Depth

connectionType

The *connectionType* variable, in Internet Explorer, indicates whether the browser is configured on a LAN or modem connection.

This variable is populated after the page code and before *doPlugins* is run.

 **Note:** This variable should only be read and never set.


You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example	Reports Affected
ct	lan or modem	lan	Traffic > Technology > Connection Type

cookiesEnabled

The `cookiesEnabled` variable indicates whether a first-party session cookie could be set by JavaScript.

This variable is populated after the page code and before `doPlugins` is run.


 **Note:** This variable should only be read and never set.

You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example
k	Y or N	Y

dc

(Deprecated) The `dc` variable lets you select the data center to which your data is sent.

 **Note:** The `dc` variable is deprecated. You should set `trackingServer` for all implementations to the value that is generated by **Code Manager** in `s_code.js`.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	112

The data center is identified in the `dc` variable in order to match **ActionSource**.

eVarN

The **eVar** variables are used for building custom reports.

When an eVar is set to a value for a visitor, the value is remembered until it expires. Any success events that a visitor encounters while the eVar value is active are counted toward the eVar value.

Max Size	Debugger Parameter	Reports Populated	Default Value
255 Bytes	V1-v75 (or v100 or v250)	Custom Conversion	""

Expiration

eVars expire after a time period you specify. After the eVar expires, it no longer receives credit for success events. eVars can also be configured to expire on success events. For example, if you have an internal promotion that

expires at the end of a visit, the internal promotion receives credit only for purchases or registrations that occur during the visit in which they were activated.

There are two ways to expire an eVar:

- You can set the eVar to expire after a specified time period or event.
- You can use force the expiration of an eVar, which is useful when repurposing a variable.

If an eVar is used in May to reflect internal promotions and expires after 21 days, and in June it is used to capture internal search keywords, then on June 1st, you should force the expiration of, or reset, the variable. Doing so will help keep internal promotion values out of June's reports.

Case Sensitivity

eVars are case insensitive, but they are displayed in the capitalization of the first occurrence. For example, if the first instance of eVar1 is set to "Logged In," but all subsequent instances are passed as "logged in," reports always show "Logged In" as the value of the eVar.

Counters

While eVars are most often used to hold string values, they may also be configured to act as counters. eVars are useful as counters when you are trying to count the number of actions a user takes before an event. For example, you may use an eVar to capture the number of internal searches before purchase. Each time a visitor searches, the eVar should contain a value of '+1.' If a visitor searches four times before a purchase, you will see an instance for each total count: 1.00, 2.00, 3.00, and 4.00. However, only the 4.00 receives credit for the purchase event (Orders and Revenue Metrics). Only positive numbers are allowed as values of an eVar counter.

Subrelations

A common requirement for a **Custom eVar** report is the ability to break down one **Custom eVar** report by another. For example, if one eVar contains gender, and another contains salary, you may ask the following question: of the female visitors to my site, how much revenue was generated by women who make more than \$50,000 per year. Any eVar that is fully sub-related allows this type of break down in reports. For example, if the gender eVar has full subrelations enabled, all other custom eVar reports can be broken down by gender, and gender can be broken down by all others. To see the relationship between two reports, only one of them needs full subrelations enabled. By default, **Campaigns**, **Products**, and **Category** reports are fully sub-related (any eVar can be broken down by campaign or products).

Syntax and Possible Values

While eVars may be renamed, they should always be referred to in the JavaScript file by eVarX, where X is a number between 1 and 75 (*or 100, or 250*).

```
s.eVarX="value"
```

When not used as a counter, eVars have the same limitations as all other variables. If the eVar is a "counter," it is expected to receive numeric values like "1" or "2.5." If more than two decimal places are given, the eVar counter rounds to two decimal places. An eVar counter may not contain negative numbers.

Examples

```
s.eVar1="logged in"
```

```
s.eVar23="internal spring promo 4"
```

Configuration Settings

eVars can be configured in the Admin Console. All eVars can be configured with a **Name**, **Type**, **Allocation**, **Expire After Setting**, or **Reset**. Each configuration setting is addressed separately.

Setting	Description
Name	Allows you to change the name of the eVar report within Analytics. The eVar should still be referenced as s.eVarX in the JavaScript code, no matter what name is given to the report in Analytics.
Type	Allows you to show whether the eVar is a Text String or Counter.
Allocation	Used to configure which value of the eVar receives credit for success events. If Allocation is set to "Most Recent (Last)," then B receives credit. If Allocation is set to "Original Value (First)" then A receives credit. If Allocation is set to "Linear", then both A and B receive credit for half the purchase value.
Expire After	Lets you determine whether an eVar expires on a specific event, like purchase, or after a custom or predefined time period.
Reset	By selecting the Reset check box for an eVar, and clicking Save at the bottom of the page, all values of that eVar are immediately expired. After this happens, only new values of the eVar receive credit for success events.

Pitfalls, Questions, and Tips

- Unlike **prop** variables, eVar variables are not allowed to be lists of delimited values. If you populate an eVar with a list of values, for example "one,two,three," then that exact string appears in reports.
- eVar counters may not contain negative numbers.

Configure Events

The *s.events* variable is used to record common shopping cart success events as well as custom success events.

Max Size	Debugger Parameter	Reports Populated	Default Value
No Limit	events	Shopping Cart Events Custom Events	N/A

An **event** should be considered a milestone within a site. Success events are most commonly populated on the final confirmation page of a process, such as a registration process or newsletter sign-up. Custom events are defined by populating the events variable with the literal values defined in the [Possible Values](#) section below.

By default, success events are configured as *counter* events. Counter events count the number of times a success event is set (x+1). Events can also be configured as *numeric* events. Numeric events allow you to specify the number to increment (as might be necessary when counting dynamic or arbitrary values, such as the number of results returned by an internal search).

A final event type, *currency*, allows you to define the amount to be added (similar to numeric events), but displays as currency in reports, and is subject to currency conversions based on the *s.currencyCode* value and the default currency setting for your report suite. For additional information on using numeric and currency events, see [Products](#).

Configuring the Variable

The **s.events** variable is enabled by default for all implementations. The seven pre-configured conversion events are automatically enabled for all new report suites. New custom events (event1-*event100* or *event1000*) can be enabled by any admin-level user using the Admin Console.

Possible Values

The following is a list of possible values for the events variable:

Event	Description	Reports Populated
prodView	Product Views	Products
scOpen	Open / Initialize a new shopping cart	Carts
scAdd	Add item(s) to the shopping cart	Cart Additions
scRemove	Remove item(s) from the shopping cart	Cart Removals
scView	View shopping cart	Cart Views
scCheckout	Beginning of the checkout process	Checkouts
purchase	Completion of a purchase (order)	Orders
event1 - event1000 (event100 for point product)	Custom events	Custom Events

Syntax and Examples

Counter Events

Counter events are set by placing the desired events in the **s.events** variable, in a comma-separated list (if multiple events are to be passed).

```
s.events="scAdd"
```

```
s.events="scAdd,event1,event7"
```

```
s.events="event5"
```

```
s.events="purchase,event10"
```

If on H23 code or higher, counter events can have integers greater than one assigned to them.

```
s.events="event1=10"
```

```
s.events="scRemove=3,event6,event2=4"
```

Implementing counter events with assigned integer values treat the event as if it fired multiple times within the image request. Counter events do not allow decimals- it is recommended to use numeric events instead if this functionality is required.

Numeric/Currency Events

Numeric and currency events must be included in the **s.events** variable, though they typically receive their numerical value (e.g., 24.99) in the **s.products** variable. This allows you to tie specific numeric and currency values to individual product entries.

With Products

```
s.events="event1"
s.products="Footwear;Running Shoes;1;99.99;event1=4.50"
```

```
s.events="event1,event4"
s.products="Footwear;Running Shoes;1;99.99;event1=4.50|event4=1.99"
```

Without Products

You can also track these types of events by leaving the product and category fields blank. This was the process to measure data such as tax and shipping before support was added for decimal values outside of the products string. If you want to pass a numeric or currency value but do not want to tie these to an individual product (such as a discount applied to an entire order, or a value captured outside of the product view or shopping cart process), this can be done by using **s.products** as shown below, but leaving the product and category fields blank.

```
s.events="event1"
s.products=";;;event1=4.50"
```

```
s.events="event1,event4"
s.products=";;;event1=4.50|event4=1.99"
```

In the Events List

Numeric/Currency Events that are given a value directly in events list apply to all products in the products list.

This is useful to track order-wide discounts, shipping, and similar values, without modifying the product price or by tracking it in the product list separately. For example, if you configured event10 to contain order-wide discounts, a purchase with a 10% discount might appear similar to the following:

```
s.events="purchase,event10=9.95"
s.products="Footwear;Running Shoes;1;69.95,Running Essentials;Socks;10;29.50"
```

On Numeric/Currency Event reports, the report total represents the de-duplicated event total (in this example, the total amount of discounts during the reporting period), not the sum of the event values for each product.



Note: if a value for a Numeric/Currency Event is specified in the products string and in the event string, the value from the event string is used.

Combining Counter Events and Numeric/Currency Events

Counter events can be passed along with numeric/currency events by passing all desired events in the **s.events** variable, and then passing only the numeric/currency events in the **s.products** variable.

Event Serialization

By default, an event is counted every time the event is set on your site.

See [Event Serialization](#) for more information.

Syntax

```
s.events="event1:3167fhjkah"
```

Examples

```
s.events="scAdd:003717174"
```

```
s.events="scAdd:user228197,event1:577247280,event7:P7fhF8571"
```

hierN

The **hierarchy** variable determines the location of a page in your site's hierarchy.

This variable is most useful for sites that have more than three levels in the site structure. For example, a media site may have 4 levels to the Sports section: Sports, Local Sports, Baseball, and Red Sox. If someone visits the Baseball page, Sports, Local Sports, and Baseball, all levels reflect that visit.

Max Size	Debugger Parameter	Reports Populated	Default Value
255 Bytes	H1-H5	Hierarchy	""

There are five **hierarchy** variables available, which must be enabled by Adobe Customer Care. At the time the hierarchy is enabled, you should decide on a delimiter for the variable and the maximum number of levels for the hierarchy. For example, if the delimiter is a comma, the sports hierarchy may display as follows.

```
s.hier1="Sports,Local Sports,Baseball"
```

Make sure that none of your section names have the delimiter in them. For example, if one of your sections is called "Coach Griffin, Jim," then you should choose a delimiter other than comma. Each hierarchy section is limited to 255 bytes, while the total variable limit is 255 bytes. After a delimiter is chosen (at the time the hierarchy is created) it is not easily changed.

Contact Adobe Customer Care about changing the delimiter for an existing hierarchy. Delimiters may also consist of multiple characters, such as || or /\, which are less likely to appear in a hierarchy section.

Syntax and Possible Values

Do not put a space between each delimiter. In the following example syntax, N is a number between one and five.

```
s.hierN="Level 1[<delimiter>Level 2[<delimiter>Level 3[...]]]"
```

Do not use the delimiter except to delimit the levels of the hierarchy. The delimiter may be any character or characters of your choice.

Examples

```
s.hier1="Toys|Boys 6+|Legos|Super Block Tub"
```

```
s.hier4="Sports/Local Sports/Baseball"
```

Configuration Settings

None

Pitfalls, Questions, and Tips


- The delimiter may not be changed after the hierarchy is set up. If the delimiter for your hierarchy must be changed, contact Adobe Customer Care.
- The number of levels may not be changed after the hierarchy is set up.

 **Note:** Changes to hierarchies can result in a service charge.

homepage

The *homepage* variable, in Internet Explorer, indicates whether the current page is set as the user's home page.

This variable is populated after the page code and before *doPlugins* is run.

 **Note:** This variable should only be read and never set.


You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example	Reports Affected
hp	Y or N	Y	Traffic > Technology > Home Page

javaEnabled

The `javaEnabled` variable indicates whether Java is enabled on the browser.

This variable is populated after the page code and before `doPlugins` is run.

 **Note:** This variable should only be read and never set.


You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example	Reports Affected
v	Y or N	Y	Traffic > Technology > Java

javascriptVersion

The `javascriptVersion` variable indicates the version of JavaScript supported by the browser.

This variable is populated after the page code and before `doPlugins` is run.

 **Note:** This variable should only be read and never set.

You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example	Reports Affected
j	1.0, 1.1, 1.2, ... 1.7	1.7	Traffic > Technology > JavaScript Version

Version H.10 and higher of the JavaScript file accurately detect up to version 1.7 (the highest version at the time H.10 was released). Prior versions of the JavaScript file only detected up to version 1.3

linkName

The `linkName` variable is an optional variable used in **Link Tracking** that determines the name of a custom, download, or exit link.

The `linkName` variable is not normally needed because the third parameter in the `tl()` function replaces it.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 bytes	pev2	File Downloads Custom Links	""

Max Size	Debugger Parameter	Reports Populated	Default Value
		Exit Links	

Custom Links refer to links that send tracking data. The *linkName* variable (or the third parameter in the *tl()* function) is used to identify the value that appears in the **Custom**, **Download**, or **Exit Links** report. If *linkName* is not populated, the URL of the link appears in the report.

Syntax and Possible Values

```
s.linkName="Link Name"
```

There are no limitations on *linkName* outside of the standard variable limitations.

Examples

```
s.linkName="Nav Bar Home Link"
```

```
s.linkName="Partner Link to A.com"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- The *linkName* variable is replaced by the third parameter in the *tl()* function.
- If the *linkName* variable and the third parameter in the *tl()* function are blank, the full URL of the link (with the exception of the query string) appears in the report (even if the link is relative).

linkType

The *linkType* variable is an optional variable used in link tracking that determines which report a link name or URL appears (custom, download, or exit links).

The *linkType* variable is not normally needed because the second parameter in the *tl()* function replaces it.

Max Size	Debugger Parameter	Reports Populated	Default Value
One character	pe=[lnk_o lnk_d lnk_e]	File Downloads Custom Links Exit Links	""

Custom links send data to Analytics. The *linkType* variable (or the second parameter in the *tl()* function) is used to identify the report in which the link name or URL appears (**Custom**, **Download**, or **Exit Links** report).

For exit and download Links, the *linkType* variable is automatically populated depending on whether the link clicked is an exit or download link. A custom link may be configured to send data to any of the three reports with this variable or with the second parameter in the *tl()* function. By setting *linkType* to 'o,' 'e,' or 'd,' the *linkName* or link URL is sent to the **Custom Links**, **Exit Links**, or **File Downloads** report respectively.

Syntax and Possible Values

The *linkType* variable syntax depends on whether you use XML or a query string.

If you are using XML, the variable may only contain a single character, namely 'o', 'e', or 'd.'

```
s.tl(this, 'o', 'Link Name');
```

If you are using the query-string `pe`, you need to use `lnk_d`, `lnk_e`, or `lnk_o`.

Examples

```
<a href="index.html" onClick="
  var s=s_gi('rsid'); **see note below on the rsid**
  s.tl(this, 'o', 'Link Name');
">My Page</a>
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- If `linkType` is not specified, custom links ('o') is assumed.

List Props

List props are a delimited list of values that are passed into a variable, then reported as individual line items. List props are most commonly implemented on pages that contain user-selectable values, such as listed items with check boxes or radio buttons. They are useful in any circumstance where you want to define multiple values in a variable without sending multiple image requests.

Considerations

- List props are enabled only on traffic variables ([props](#)).
- Pathing and correlations cannot be enabled for list props.
- Analytics provides visits and unique visitors to almost every report, including all list prop reports.
- Classifications are supported for list props.
- Any custom traffic variable can become a list prop. (Exceptions: [pageName](#), [channel](#), and [server](#).)
- When defining duplicate values in the same image request, instances are deduplicated. Identical values appearing multiple times in the same hit are only counted once.

A prop can be changed into a list prop on the Admin Tools > Report Suite > Traffic Variables page by enabling List Support and then selecting a delimiter. Popular delimiters are colons, semi-colons, commas, or pipes. The delimiter can technically be any of the first 127 ASCII characters.

Implementation Examples

When you request enabling of list props, indicate the delimiter that you would like to use. After the `s.prop` of your choice is enabled, multiple values can be set in the variable as shown in the following examples:

A list prop delimited by a pipe, passing in two values:

```
s.prop1="Banner ad impression|Sidebar impression"
```

A list prop delimited by a comma passing in several values:

```
s.prop2="cerulean,vermillion,saffron"
```

List props can also be sent with a single value:

```
s.prop3="Single value"
```

The delimiter can be changed at any time. However, the implementation must match the new delimiter. Failure to use the correct delimiter results in the list prop value being treated as a single concatenated line item in reporting.

Because a list prop is still a Traffic Variable, it is subject to Traffic Variable limitations. List props are limited to 100 bytes of data and are affected by case sensitivity settings.

List Variable

Also known as List Var. Similar to how List Props function, List Vars allow multiple values within the same image request. They also act similarly to eVars, which persist beyond the image request they were defined on. You can use these variables to see cause and effect among multiple elements on a single page, such as product lists, wish lists, lists of search refinements, or lists of display ads.

Considerations:

- List Vars remember their specific values by referencing the VisitorID cookie in the visitor's browser.
- A limit of 250 maximum values are stored at one time per visitor. If 250 values per visitor are exceeded, the latest 250 values are used. Expiration for these values is based on the configured expiration for the variable.
- Each delimited value can contain a maximum of 255 characters (or less if using multi-byte characters). This is the maximum length of each element.
- There is no limit to the number of characters within this variable. The only exception to this limitation is within older Internet Explorer browsers, which impose a 2083-character limitation on all URL requests.
- A total of three List Vars are available per report suite.
- Using List Vars requires H23 code or higher.
- List Vars can be classified.
- If duplicate values are defined in the same image request, list vars deduplicate all instances of those values.
- The most granular list vars can be segmented is on a hit (or page view) level. If you have a list var with three values in the same image request, any segment rules that match one value will pull all three into reporting. Conversely, if an exclude rule is defined that matches a single value, all three values are excluded.

Configuration

You can access the configuration in the Admin Console and update it without Adobe Client Care having to get involved:

1. Go to **Analytics > Admin > Report Suites**
2. Select the report suite.
3. Click **Edit Settings > Conversion > List Variables**.

- **Name:** Each delimited value can contain a maximum of 255 characters (or less if using multi-byte characters). This is the maximum length of each element.
- **Value Delimiter:** The character used to separate values within the List Var. Most commonly these are characters such as commas, colons, pipes, or something similar.



Note: Multi-byte characters are not supported as delimiters in List Vars. The delimiter must be single byte.

- **Expiration:** Similar to eVar expiration, this determines the amount of time that can occur between the List Var and the conversion event for them to be related.
 - **At a page view or visit level:** Success events beyond the page view or visit would not link back to any values within the List Var.
 - **Based on a time period, such as day, week, month, etc:** Success events beyond the specified time period would not link back to any values within the List Var. A custom number of days can be defined as well.
 - **Specific conversion events:** Any other success events that fire after the specific event designated would not link back to any values within the List Var.
 - **Never:** Any amount of time can pass between the List Var and success event.
- **Allocation:** This setting determines how success events divide credit between values:

- **Full:** All variable values defined prior to the variable's expiration get full credit for success events.
- **Linear:** All variable values defined prior to the variable's expiration get credit divided credit for conversion events.
- Variable values are never overwritten, but instead added to the values that get credit for success events.
- **Max Values:** Designates the number of active values allowed for this list variable. For example, if set to 3, only the last 3 values captured is saved and any previous values captured are discarded. Note that if multiple values for the same list var are sent in on the same hit and you have restricted using max values, each value will have the same timestamp and there is not guarantee as to which value is saved.

A limit of 250 maximum values are stored at one time per visitor. If 250 values per visitor are exceeded, the latest 250 values are used. Expiration for these values is based on the configured expiration for the variable.

The Max Values setting is useful to limit attribution to a specific number of values. For example, if a list var is set to "A,B,C" on the first page of a visit, then set to "X,Y,Z" on the next page, attribution is distributed to these six values based on the allocation. If you wanted to limit attribution to only "X,Y,Z", you can set max values to three.

To set up or edit List Vars, go to **Analytics > Admin > Report Suites > Edit Settings > Conversion > List Variables**.

Implementation Examples

Each of the following examples use a comma for the value delimiter.

Defining a single value within a List Var:

```
s.list1="Cat";
```

Passing in multiple values:

```
s.list2="Tabby,Persian,Siamese";
s.list1="Product 1,Product 2,Product 3";
```

Attributing revenue to a List Var:

```
//Define this code on the landing page:
s.list3="Top Banner Ad,Side Bar Ad,Internal Campaign 1";

//Have these variables fire on the purchase confirmation page:
s.products=";Kitten;1;50"
s.events="purchase";
```

This result would show three line items with \$50 each in revenue. (Top Banner Ad; Side Bar Ad; and Internal Campaign 1.) Note the total for this report deduplicates revenue, so the total would also reflect \$50.

Attributing revenue to a List Var that was set multiple times during a visit:

Allocation: Full

Expiration: Visit

Page	s.list1	s.events/s.products
Page 1	s.list1="value1,value2,value3";	(not set)
Page 2	s.list1="value4,value5,value6";	s.events="purchase"; s.products=";product;1;200"

Result: All values set in the list var1 at any point during the visit (value1,value2,value3,value4,value5,value6) get full credit for the purchase.

maxDelay

The `s.maxDelay` variable is used primarily in Genesis DFA integrations to determine the timeout period in contacting the DFA host. If Adobe does not receive a response from DFA's servers within the specified period set in the `s.maxDelay` variable, the connection is severed, and data is processed normally. Implement this variable if you are concerned with DFA's response time on each page. It is recommended to experiment with this value to determine the optimum timeout period.

Implementation Example

```
s.maxDelay="750";
```

Properties

- This variable is an optional event metric populated via the JavaScript implemented on your site.
- If the DFA host does not respond within the given amount of time, the event designated to Timeout runs (assigned via the Genesis integration wizard).
- This variable can only contain a numeric value.
- The amount of time specified is measured in milliseconds.
- Increasing the wait time collects more DFA data, but also increases the risk of losing Analytics hit data.

Losing Analytics hit data would occur when the user navigates away from the page during the `s.maxDelay` period.

- Decreasing the wait time will lower the risk of losing Analytics hit data, but can reduce the amount of DFA data sent with hit data.

Losing DFA integration data would occur when the `s.maxDelay` period does not accommodate enough time for the DFA host to respond.



Note: Adobe does not have control over DFA's response time. If you are seeing consistent issues even after raising the max delay period to a reasonable time frame, consult your organization's DFA account administrator.

mediaLength

The `mediaLength` variable specifies the total length of the media being played.

Max Size	Debugger Parameter	Reports Populated	Default Value
No max size for entire pev3 request - size is limited to the browser's URL length limit.	pev3	Time Spent on Video; Video Segments Viewed	None

Syntax and Possible Values

autoTrack Method:

If using `s.Media.autoTrack`, the `mediaLength` variable does not need to be implemented explicitly. It is determined automatically by the AppMeasurement for JavaScript code.

Manual Tracking Method:

Syntax:

```
s.Media.open(mediaName,mediaLength,mediaPlayerName)
```

Possible Values:

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
```

Examples

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
```

Resulting pev3 parameter syntax: pev3= [Asset Name]---[Total length of asset]---[Player name]---[Total seconds consumed]---[Timestamp]---[Chronological record of all starts and stops along with accompanying markers]

```
Possible pev3 values: pev3=de_bofr_1045Making_400k---414---Windows Media Player
11.0.5721.5230---288---1207893838---S0E0S0E256S0E32
```

Pitfalls, Questions, and Tips

- You must call the media tracking methods only if the player cannot be tracked using **s.Media.autoTrack = true**.
- If not tracking using **autoTrack**, be sure to set the length in seconds.

mediaName

This variable specifies the name of the video or media item.

It is only available via the **Data Insertion API** and **Full Processing Data Source**.

Max Size	Debugger Parameter	Reports Populated	Default Value
64 KB	pev3	Videos; Next Video Flow; Previous Video Flow; Video Segments Viewed; Time Spent on Video	None

Syntax and Possible Values

autoTrack Method:

If using **s.Media.autoTrack**, the *mediaName* variable does not need to be implemented explicitly. It is determined automatically by the AppMeasurement for JavaScript code.

Manual Tracking Method:

Syntax:

```
s.Media.open(mediaName,mediaLength,mediaPlayerName)
```

```
s.Media.play(mediaName,mediaOffset)
```

```
s.Media.stop(mediaName,mediaOffset)
```

```
s.Media.close(mediaName)
```

Possible Values:

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
```

```
s.Media.play("de_bofr_1045Making_400k", "0")
```

```
s.Media.play("de_bofr_1045Making_400k", "414")
```

```
s.Media.close("de_bofr_1045Making_400k")
```

Examples

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
s.Media.play("de_bofr_1045Making_400k", "0")
s.Media.play("de_bofr_1045Making_400k", "414")
s.Media.close("de_bofr_1045Making_400k")
```

Resulting pev3 parameter syntax: pev3=[Asset Name]--*--[Total length of asset]--*--[Player name]--*--[Total seconds consumed]--*--[Timestamp]--*--[Chronological record of all starts and stops along with accompanying markers]

Possible pev3 Values:

```
pev3=de_bofr_1045Making_400k--*--414--*--Windows Media Player
11.0.5721.5230--*--288--*--1207893838--*--S0E0S0E256S0E32
```

Pitfalls, Questions, and Tips

- You must call the media tracking methods only if player cannot be tracked using `s.Media.autoTrack = true`.
- This variable is stored as a mySQL TEXT variable as opposed to VARCHAR(100).

mediaPlayer

This variable specifies the player used to consume a video or media item.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 Bytes	pev3	Video Players	None

Syntax and Possible Values

autoTrack Method:

```
s.Media.playerName = "My Custom Player Name" //configure player name in global JavaScript or
ActionSource
```

Manual Tracking Method:

```
s.Media.open(mediaName,mediaLength,mediaPlayerName)
```

Possible Values:

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
```

Examples

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
```

Resulting pev3 parameter syntax: pev3=[Asset Name]--*--[Total length of asset]--*--[Player name]--*--[Total seconds consumed]--*--[Timestamp]--*--[Chronological record of all starts and stops along with accompanying markers]

Possible pev3 Values: pev3=de_bofr_1045Making_400k--*--414--*--Windows Media Player
11.0.5721.5230--*--288--*--1207893838--*--S0E0S0E256S0E32

Pitfalls, Questions, and Tips

You must call the media tracking methods only if player cannot be tracked using `s.Media.autoTrack = true`.

mediaSession

This variable specifies the segments of a video or media asset consumed.

Max Size	Debugger Parameter	Reports Populated	Default Value
255 Bytes	pev3	Time Spent on Video Video Segments Viewed	None

Syntax and Possible Values

autoTrack Method:

If using **s.Media.autoTrack**, the *mediaName* does not need to be implemented explicitly. It will be determined automatically by the AppMeasurement for JavaScript code.

Manual Tracking Method:

Syntax:

```
s.Media.open(mediaName,mediaLength,mediaPlayerName)
```

```
s.Media.play(mediaName,mediaOffset)
```

```
s.Media.stop(mediaName,mediaOffset)
```

Possible Values:

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
```

```
s.Media.play("de_bofr_1045Making_400k", "0")
```

```
s.Media.play("de_bofr_1045Making_400k", "414")
```

Examples

```
s.Media.open("de_bofr_1045Making_400k", "414", "Windows Media Player 11.0.5721.5230")
```

```
s.Media.play("de_bofr_1045Making_400k", "0")
```

```
s.Media.play("de_bofr_1045Making_400k", "414")
```

Resulting pev3 parameter syntax: pev3=[Asset Name]--*--[Total length of asset]--*--[Player name]--*--[Total seconds consumed]--*--[Timestamp]--*--[Chronological record of all starts and stops along with accompanying markers]

Possible pev3 Values: pev3=de_bofr_1045Making_400k--*--414--*--Windows Media Player 11.0.5721.5230--*--288--*--1207893838--*--S0E0S0E256S0E32

Pitfalls, Questions, and Tips

You must call the media tracking methods only if player cannot be tracked using **s.Media.autoTrack = true**.

Media.trackEvents

The *Media.trackEvents* variable identifies which events should be sent with a media hit.

It is only applicable with JavaScript and **ActionSource**.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	s.Media.trackEvents="None"

Syntax and Possible Values

Event names such as event1 or purchase.

Examples

```
s.Media.trackEvents="event1,purchase"
```

Pitfalls, Questions, and Tips

Make sure to populate **trackVars** with "events" whenever this variable is populated.

Media.trackVars

The *Media.trackVars* variable identifies which variables should be sent with a media hit.

It is only applicable with JavaScript and **ActionSource**.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	s.Media.trackVars="None"

Syntax and Possible Values

Variable names such as **propN**, *eVarN*, *events*, *channel*, and so forth.

Examples

```
s.Media.trackVars="prop2,events,eVar3"
```

Pitfalls, Questions, and Tips

- Even if eVar3 is specified in **trackVars**, it is sent with the media hit.

mobile

The *mobile* variable controls the order in which cookies and subscriber IDs are used to identify visitors.

See [Mobile Tracking over WAP and I-Mode Protocols](#).

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	/5/ or /1/ in path of image url	N/A	None

Syntax and Possible Values

```
s.mobile="any_string" //subscriber id used first, produces /5/ in path of image url
s.mobile="" // if set to an empty string or not set at all, cookies used first, produces /1/
in path of image url
```

Pitfalls, Questions, and Tips

Use cross-visitor identification to mitigate possible spikes in visitor traffic when using the *s.mobile* variable with the JavaScript cookie implementation.

pageName

The *pageName* variable contains the name of each page on your site.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 bytes	pageName	Pages Paths	page URL

The *pageName* variable should be populated with a value that business users recognize. In most cases the *pageName* value is not the URL or the path to the file. Common *pageName* values include names such as "Home Page," "Checkout," "Purchase Thank you," or "Registration."

Be careful not to allow new-line, -em or -en dashes, or any HTML characters to appear in the page name and other variables. Some browsers send new line characters while others don't, which causes the data in Analytics to be split between two seemingly identical page names. Many word processors and email clients will automatically convert a hyphen into an -en or -em dash when typing. Since -en and -em dashes are illegal characters in Analytics variables (ASCII characters with codes above 127), Analytics won't record the page name containing the illegal character and show the URL instead.

If *pageName* is left blank, the URL is used to represent the page name. Leaving *pageName* blank is often problematic because the URL may not always be the same for a page (www.mysite.com and mysite.com are the same page with different URLs).

Syntax and Possible Values

The *pageName* variable should contain a useful identifier for business users of Analytics.

```
s.pageName="page_name"
```

There are no limitations on *pageName* outside of the standard variable limitations.

Examples

```
s.pageName="Search Results"
```

```
s.pageName="Standard Offer List"
```

Configuration Settings

Administrators have the ability to change the visible page name in Analytics with the **Name Pages** tool, which is potentially dangerous and may negatively affect your reports. Please contact Adobe Customer Care before using the **Name Pages** tool.

Pitfalls, Questions, and Tips

Make sure the *pageName* doesn't contain illegal characters.

pageType

The *pageType* variable is used only to designate a 404 Page Not Found Error page.

Max Size	Debugger Parameter	Reports Populated	Default Value
20 bytes	pageType	Paths > Pages > Pages Not Found	""

The *pageType* variable captures the errant URL when a 404 Error page is displayed, which allows you to quickly find broken links and paths that are no longer valid on the custom site. Set up the *pageType* variable on the error page exactly as shown below.

Do not use the page name variable on 404 error pages. The *pageType* variable is only used for the 404 Error page.

In most cases, the 404 Error page is a static page that is hard-coded. In these cases, it is important that the reference to the .JS file is set to an appropriate global or relative path/directory.

Syntax and Possible Values

The only allowable value of `pageType` is "errorPage" as shown below.

```
s.pageType="errorPage"
```

Examples

```
s.pageType="errorPage"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

To capture other server-side errors (such as 500 errors), use a prop to capture the error message and put "500 Error: <URL>" where <URL> is the URL requested, in the `pageName` variable. By following this course of action, you can use **Pathing** reports to see which paths caused users to generate 500 errors. The prop explains which error message is given by the server.

pageURL

The `pageURL` variable overrides the actual URL of the page.

In rare cases, the URL of the page is not the URL that you would like reported in Analytics.

Max Size	Debugger Parameter	Reports Populated	Default Value
No Limit*	G	Traffic > Segmentation > Most Popular Pages Paths	Page URL



Note: Although Adobe allows `pageURL` values up to 64k, some browsers impose a size limit on the URL of image requests. To prevent truncation of other data, page URLs longer than 255 bytes are split, with the first 255 bytes appearing in the `g=` parameter, with the remaining bytes appearing later in the query sting in the `-g=` query parameter.

Syntax and Possible Values

The `pageURL` variable must be a valid URL, with a valid protocol. The domain will be forced to display in lower-case before being populated in reports, and the query string may be stripped, depending on Analytics settings.

```
s.pageURL="proto://domain/path?query_string"
```

Only URL-compatible characters are allowed as the page URL.



Note: It is strongly advised that you contact your Adobe consultant or Customer Care before using the `pageURL` variable for custom purposes.

Examples

```
s.pageURL="http://mysite.com/home.jsp?id=1224"
```

```
s.pageURL="http://www.mysite.com/"
```


Configuration Settings

None

plugins

The *plugins* variable, in Netscape and Mozilla-based browsers, lists the plugins installed on the browser.

This variable is populated after the page code and before *doPlugins* is run.

 **Note:** This variable should only be read and never set.

You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example	Reports Affected
p	Recognized plugins	IE Tab Plug-in;QuickTime Plug-in 7.1.6;Mozilla Default Plug-in;iTunes Application Detector;Adobe Acrobat;ActiveTouch General Plugin Container;Shockwave Flash;Microsoft Office 2003;Java(TM) Platform SE 6 U1;Windows Media Player Plug-in Dynamic Link Library;Microsoft® DRM;	Traffic > Technology > Plugins

products

The *products* variable is used for tracking products and product categories as well as purchase quantity and purchase price. Products is typically set in conjunction with a cart event or a *prodView* event.

Important:

In January, 2016, Adobe updated the logic that sets the prodView event automatically, which happens when there is a product but no event. This update may cause an increase in prodView events. prodViews will increase only when:

- 1. The events variable contains nothing but an unrecognized event, such as shoppingCart or cart, which are not valid events.*
- 2. The products variable is not empty.*

A possible side effect is that merchandising eVars triggered by prodView events could be associated with an empty product, but only if the product list contains only an invalid product (such as a semicolon with no product listed).

The *products* variable tracks how users interact with products on your site. For instance, the products variable can track how many times a product is viewed, added to the shopping cart, checked out, and purchased. It can also track the relative effectiveness of merchandising categories on your site. The scenarios below are common for using the products variable.

The *products* variable should always be set in conjunction with a success event.

Max Size	Debugger Parameter	Reports Populated	Default Value
The " Product " and " Category " sections of products each have a limit of 100 bytes. Adobe does not impose a size limit on products, though most browsers impose a size limit on the total URL length of image requests.	products	Products Categories (optional) Revenue (optional) Units (optional) Custom Events (optional) eVars (optional)	" "

Syntax of the Products Variable

```
"Category;Product;Quantity;Price;eventN=X[ |eventN2=X2];eVarN=merch_category[ |eVarN2=merch_category2]"
```

Field	Definition
Category	Contains the associated product category. In version 15, products can be associated with multiple categories which fixes a limitation present in version 14. If you were previously not recording a product category, you are encouraged to start populating this field for report suites that are on version 15.
Product	(Required) The identifier used to track a product. This identifier is used to populate the Products report. Be sure to use the same identifier through the checkout process.
Quantity	The number of units purchased. This field must be set with a purchase event to be recorded.
Price	Refers to the combined cost of the total quantity purchased (units x individual unit price), not to the individual price. This field must be set with a purchase event to be recorded.
Events	Currency events associated with the specified product. See Product-Specific Currency Events and Order-Wide Currency Events .
eVars	Merchandising eVar values associated with a specific product. See Merchandising Variables .

The values included in the *products* variable are based on the type of event you are recording. The category/product delimiter (;) is required as a place holder when omitting Category. Other delimiters are required only if they are necessary to distinguish which parameter you are including, as shown in the examples on this page.

Setting products with Non-Purchase Events

The *products* variable must be set in conjunction with a success event.

Syntax

```
s.events="prodView"  
s.products="Category;Product[ ,Category;Product ]"
```

In the examples below, product attributes (category) are separated by semicolons. Multiple products are separated by commas.

Example 1: Single Product

```
s.events="prodView"  
s.products="Footwear;Running Shoes"
```

Example 2: Multiple Products

```
s.events="prodView"  
s.products="Footwear;Running Shoes,;Running Socks"
```

Setting products with a Purchase Event

The *purchase* event should be set on the final confirmation ("Thank You!") page of the order process. The product name, category, quantity, and price are all captured in the *products* variable. Although the *purchaseID* variable is not required, it is strongly recommended in order to prevent duplicate orders.

Syntax

```
s.events="purchase"
s.products="Category;Product;Quantity;Price[,Category;Product;Quantity;Price]"
s.purchaseID="1234567890"
```

Example 1: Single Product

```
s.events="purchase"
s.products="Footwear;Running Shoes;1;69.95"
s.purchaseID="1234567890"
```

Example 2: Multiple Products

```
s.events="purchase"
s.products="Footwear;Running Shoes;1;69.95,Running Essentials;Running Socks;10;29.99"
s.purchaseID="1234567890"
```



Note: Price refers to the total price (unit price x units). For instance, 3 widgets purchased at 19.99 each would equal 59.97 (such as ";Widget;3;59.97").

Product-Specific Currency Events

If a currency event receives a value in the *products* variable instead of the events variable, it applies only to that value. This is useful to track product-specific discounts, product shipping, and similar values. For example, if you configured event 1 to track product shipping, a product with a "4.50" shipping charge might appear similar to the following:

```
s.events="event1"
s.products="Footwear;Running Shoes;1;99.99;event1=4.50"
```

In this example, the value of 4.50 is associated directly with the "Running Shoes" product. If you add event1 to the products report, you'll see "4.50" listed for the "Running Shoes" line item. Similar to Price, this value should reflect the total for the quantity listed. If you have 2 items with a 4.50 shipping charge each, event1 should be "9.00".

Example 1: Single Numeric/Currency Event

```
s.events="purchase,event1"
s.products="Footwear;Running Shoes;1;69.95;event1=7.59"
```

Example 2: Multiple Numeric/Currency Events

```
s.events="purchase,event1,event2"
s.products="Footwear;Running Shoes;1;69.95;event1=7.59|event2=19.45"
```

Order-Wide Currency Events

If a currency event receives a value in the events list instead of the *products* variable, it applies to all products in the *products* variable. This is useful to track order-wide discounts, shipping, and similar values, without modifying the product price or by tracking it in the product list separately.

For example, if you configured event10 to contain order-wide discounts, a purchase with a 10% discount might appear similar to the following:

```
s.events="purchase,event10=9.95"
s.products="Footwear;Running Shoes;1;69.95,Running Essentials;Running Socks;10;29.50"
s.purchaseID="1234567890"
```

On currency event reports, the report total represents the de-duplicated event total (in this example, the total amount of discounts during the reporting period), not the sum of the event values for each product. For example, you would see "9.95" listed for both "Running Shoes" and "Running Socks", and the total would also be "9.95".



Note: if a value for the same Numeric/Currency Event is specified in the products variable and in the events variable, the value from the events is used.

Setting products with Merchandising eVars

See [Merchandising Variables](#).

Pitfalls, Questions, and Tips

- The *products* variable should always be set in conjunction with a **success** event (events). If no **success** event is specified, the default event is **prodView**.
- Strip all commas and semicolons from product and category names before populating products.
- Strip all HTML characters (registered symbols, trademarks, and so forth).
- Strip currency symbols (\$) from the price.

Examples

<code>s.products="Category;ABC123"</code>
<code>s.products="Category2;ABC123,;ABC456"</code>
<code>s.products="Category3;ABC123;1;10"</code>
<code>s.products="Category;ABC123;1;10,;ABC456;2;19.98"</code>
<code>s.events="event1"</code> <code>s.products="Category;ABC123;;event1=1.99"</code>
<code>s.events="event1"</code> <code>s.products="Category;ABC123;1;10;event1=1.99"</code>
<code>s.events="event1"</code> <code>s.products="Category;ABC123;1;10;event1=1.99,;ABC123;2;19.98;event1=1.99"</code>
<code>s.events="event1,event2"</code> <code>s.products="Category;ABC123;1;10;event1=1.99 event2=25"</code>
<code>s.events="event1,event2"</code> <code>s.products="Category;ABC123;1;10;event1=1.99 event2=25;evar1=2 Day Shipping"</code>
<code>s.events="event1,event2"</code> <code>s.products="Category;ABC123;1;10;event1=1.99 event2=25;evar1=2 Day Shipping evar2=3 Stars"</code>
<code>s.events="event1,event2"</code> <code>s.products="Category;ABC123;1;10;event1=1.99 event2=25;evar1=2 Day Shipping,;ABC456;2;19.98;event1=1.99 event2=100;evar1=Ground Shipping"</code>

```
s.events="event1,event2,event3"

s.products="Category;ABC123;1;10;event1=1.99|event2=25;evar1=2 Day
Shipping,;ABC456;2;19.98;event1=1.99|event2=100;evar1=Ground Shipping,;;;event3=2.9;evar3=20%
off"

s.events="event1,event2,event3=9.95"

s.products="Category;ABC123,;ABC456;2;19.98;event1=1.99|event2=100;evar1=Ground
Shipping,;;;event3=2.9;evar3=20% off"
```

propN

Property (**prop**) variables are used for building custom reports within the **Traffic Module**.

The props variable may be used as counters (to count the number of times a page view is sent), for pathing reports, or in correlation reports.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 bytes	c1-c75	Custom Traffic	""

Syntax and Possible Values

```
s.propN="value"
```

There are no limitations on **property** variables outside of the standard variable limitations.

Examples

```
s.prop2="editorial"
```

```
s.prop15="toy category"
```

Configuration Settings

Contact Adobe Customer Care about showing **Visit**, **Visitor**, and **Path** metrics for **prop** variables.

purchaseID

The *purchaseID* is used to keep an order from being counted multiple times in reporting.

Whenever the **purchase** event is used on your site, you should use the *purchaseID* variable.

Max Size	Debugger Parameter	Reports Populated	Default Value
20 bytes	purchaseID	Conversion > Purchases > Revenue Conversion	""

When a visitor purchases an item from your site, *purchaseID* is populated on the "Thank You" page at the same place the **purchase** event is fired. If the *purchaseID* is populated, the products on the "Thank You" page are counted only once per *purchaseID*. This is critical because many visitors to your site will save the "Thank You" or "Confirmation Page" for their own purposes. The *purchaseID* keeps purchases from being counted each time the page is viewed.

In addition to keeping the purchase data from being counted twice, the *purchaseID*, when used, keeps all conversion data from being double counted in reports.

Syntax and Possible Values

```
s.purchaseID="unique_id"
```

The *purchaseID* must be 20 characters or fewer, and be standard ASCII.

Examples

```
s.purchaseID="11223344"
s.purchaseID="a8g784hjqlmnp3"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

The *purchaseID* variable allows all conversion variables on the page to be counted only once in reports.

referrer

The *referrer* variable can be used to restore lost referrer information.

Server-side and JavaScript redirects are often used to route visitors to proper locations. However, when a browser is redirected, the original referring URL is lost.

Max Size	Debugger Parameter	Reports Populated	Default Value
255 bytes	R	Traffic > Finding Methods Conversion > Finding Methods	document.referrer

Many companies use redirects in many places throughout their websites. For example, a visitor may be sent through a redirect from a search engine paid search result. When a browser is redirected, the referrer is often lost. The *referrer* variable may be used to restore the original *referrer* value on the first page after a redirect. The *referrer* may be populated server-side, or via JavaScript from the query string.

For Analytics to record a referrer, it must be "well formed," meaning that it must follow the standard URL format, with a protocol and appropriate location.

Syntax and Possible Values

```
s.referrer="URL"
```

Only URL-compatible values should be in the *referrer*. Make sure the string is URL encoded (no spaces).

Examples

```
s.referrer="http://www.google.com/search?q=search+string"
s.referrer=<%=referrerVar%> // populated server-side
if(s.getQueryParam('ref'))
s.referrer=s.getQueryParam('ref')
```

Configuration Settings

None


Pitfalls, Questions, and Tips

The *referrer* must look like a standard URL and include a protocol.

resolution

The *resolution* variable indicates the monitor resolution of the visitor viewing the web page.

This variable is populated after the page code and before *doPlugins* is run.

 **Note:** *This variable should only be read and never set.*

You may read these values and copy them into `props/eVars`, but you should never alter them. This variable is introduced with version H.11 of the JavaScript file.

Query Param	Value	Example	Reports Affected
s	WxH	1680x1050	Traffic > Technology > Monitor Resolution

s_objectID

The *s_objectID* variable is a global variable that should be set in the **onClick** event of a link.

By creating a unique object ID for a link or link location on a page, you can either improve visitor activity tracking or use Activity Map to report on a link type or location, rather than the link URL.

 **Note:** *A trailing semicolon (;) is required when using s_objectID with [Activity Map](#).*

Max Size	Debugger Parameter	Reports Populated	Default Value
100 Bytes	OID	Activity Map, ClickMap	The Absolute URL of a Clicked Link

There are three common reasons to use *s_objectID*:

- To aggregate visitor activity that changes often during a day.
- To separate link activity that Activity Map combines.
- To improve the accuracy of Activity Map data reporting.

Aggregate Clicks on Highly Dynamic Links

If your site is highly dynamic, and links on some pages change throughout the day, *s_objectID* may be used to identify the location of a link on the page. If *s_objectID* is set to "top left 1" or "top left 2," which represents the first link in the top left of the page for example, then all links that appear in that location (or that have *s_objectID* set to the same value) are reported together with visitor click map. If you don't use *s_objectID*, you see the number of times that a specific link was clicked, but you lose insight into how all the other links in that location were used by visitors to your site.

Separate Clicks Combined

If the *pageName* variable on your site is used to show the section or template a visitor is viewing, rather than the specific page the visitor is viewing, you may want to use *s_objectID* to separate links that appear on multiple versions of that page template. For example, if you have a template page for all products on your site, it is likely that there is a link to your home page and to a search box from that template on all pages. If you want to see how those links are used on an individual product basis (rather than a template basis), you can populate *s_objectID* with a product

specific value such as "prod 123789 home page" or "prod 123789 search." Once completed, Activity Map reports on those links at an individual product basis.

Improving Activity Map Accuracy

In some cases, browsers other than Internet Explorer, Firefox, Netscape, Opera, and Safari are not reported. Although this is a small percentage, it accounts for some clicks and other metrics. Use *s_objectID* within links to uniquely identify the addresses the browser reporting issue. The following is an example of how to update your links to use *s_objectID*:

```
<a href="/art.jsp?id=559" onClick="s_objectID='top left 1';">Article 559</a>
<a href="/home.jsp" onClick="s_objectID='prod 123789 home page';">Home</a>
```

Syntax and Possible Values

s_objectID may contain any text identifier.

```
s_objectID="unique_id"
```

There are no limitations on *s_objectID* outside of the standard variable limitations.

Examples

```
s_objectID="top left 2"
```

```
s_objectID="prod 123789 search"
```

Configuration Settings

None

server

The *server* variable is used to show either the domain of a web page (to show which domains people come to) or the server serving the page (for a load balancing quick reference).

Max Size	Debugger Parameter	Reports Populated	Default Value
100 bytes	server	Servers	""

If your site has more than one domain serving the same content, the *server* variable can be used to track which of those domains visitors are using. The following JavaScript will populate the domain of the page into the *server* variable.

```
s.server=window.location.hostname
```

If you are using the *server* variable to give a quick guide to load balancing, you could put a server name or number into the *server* variable. See the following example:

```
s.server="server 14"
```

While the **Most Popular Servers** report may be used as a load balancing quick reference, it is not a precise measure of server load. For example, back-button traffic does not increase server load, but is shown in reports. The report does not show which servers are serving images or large downloads.

Syntax and Possible Values

```
s.server="server_name"
```

There are no limitations on the *server* variable outside of the standard variable limitations.

Examples

```
s.server="server 18"
s.server=window.location.hostname
```

Configuration Settings

None

Pitfalls, Questions, and Tips

The *server* variable can be used to show which domains are most popular or which servers are serving the most pages.

state

The *state* and *zip* variables are conversion variables.

They are like eVars in that they capture events, but unlike eVars, they don't persist. The *zip* and *state* variables are like eVars that expire immediately.

Max Size	Debugger Parameter	Reports Populated	Default Value
50 bytes	state	Conversion > Visitor Profile > Visitor State	""

Because the *state* and *zip* variables expire immediately, the only events associated with them are events that are fired on the same page on which they are populated. For example, if you are using *state* to compare conversion rates by state, you should populate the *state* variable on every page of the checkout process. For conversion sites, Adobe recommends using the billing address as the source for the Zip Code, but you may choose to use the shipping address instead (assuming there is only one shipping address for the order). A media site may choose to use *zip* and *state* for registration or ad click-through tracking.

Syntax and Possible Values

```
s.state="state"
```

The *state* variable does not impose any special value or format restrictions. There are no limitations on *state* outside of the standard variable limitations.

Examples

```
s.state="california"
```

```
s.state="prince edward island"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Populate *state* on every page that a relevant event is fired (such as each page of the checkout process).
- The *zip* and *state* variables act like eVars that expire on the Page View.

timestamp

This variable lets you customize the timestamp of a hit similar to the AppMeasurement libraries for other platforms.

Max Size	Debugger Parameter	Reports Populated	Default Value
4 bytes	Date/Time	Not reported directly.	Set by data collection servers.

Syntax

```
s.timestamp="UNIX or ISO-8601 format timestamp"
```

The *timestamp* variable must be in the format explained in the next section.



Important: Your report suite must be timestamp-enabled by Customer Care before you can use the *timestamp* variable. After timestamp support is enabled, all hits sent to this report suite from JavaScript must have a timestamp manually set (using *s.timestamp*) or the hits will not be recorded.

Additionally, if you enable timestamp support on a report suite to support offline tracking, all hits sent to this report suite from JavaScript must also have a timestamp manually set (using *s.timestamp*). You cannot send both time-stamped and non-time-stamped hits to the same report suite.

You can also use the [Timestamps Optional](#) setting to mix timestamped and non-timestamped data in the same global report suite, send timestamped data from a mobile app to a global report suite, and upgrade apps to employ timestamps without having to create a new report suite.

Timestamp Formats

Timestamps must be in UNIX (seconds since Jan 1st 1970) or ISO-8601 format, with the following restrictions on the accepted ISO-8601 format:

- Both date and time must be provided, separated by "T"
- The date must be a calendar date with full precision (year, month, and day). . Week dates and ordinal dates are not supported.
- The date can be in standard or extended format (YYYY-MM-DD or YYYYMMDD), but they must include the hour and minute. Seconds are optional (HH:MM, HH:MM:SS, HHMM, or HHMMSS). Fractional minutes and seconds can be passed in, but the fractional part is ignored.
- An optional time zone can be specified in standard or extended format (±HH, ±HH:MM, ±HH, ±HHMM, or Z)

UNIX timestamps continue to be supported (seconds since Jan 1st 1970).

Examples

```
s.timestamp=Math.round((new Date()).getTime()/1000);
```

```
s.timestamp="2012-04-20T12:49:31-0700";
```

The following list contains examples of valid ISO-8601 format timestamps:

```
2013-01-01T12:30:05+06:00
2013-01-01T12:30:05Z
2013-01-01T12:30:05
2013-01-01T12:30
```

Configuration Settings

A report suite must be enabled to accept custom timestamps by Customer Care before you can use this variable. After custom timestamps are enabled, all hits sent to the report suite must contain a timestamp or they are discarded.

Pitfalls, Questions, and Tips

- Timestamps are primarily used to track offline data on mobile platforms. Custom timestamps are typically disabled unless you are collecting both web and offline app data in the same report suite.
- Data is timestamped when offline data is enabled in the mobile SDK (default setting) or anytime a report suite is configured to accept time-stamped data. Data collected offline on mobile devices may be sent hours or weeks after the date when it happened. These hits may be queued within the Analytics platform for minutes or hours longer than hits without timestamps:
 - For time-stamped data sent in very near current time, the probable delay is 10-15 minutes.
 - For time-stamped data sent in from yesterday, the probable delay is about 2 hours.
 - For time-stamped data sent in that is older than yesterday, every day adds about 1 hour of delay, up to 15 days ago, when the delay stops going up.
- Timestamp-enabled session data is kept for up to 92 days.

trackingServer

The *trackingServer* variable is used for first-party cookie implementation to specify the domain at which the image request and cookie is written.

Used for non-secure pages. If *trackingServer* is defined, nothing goes to 2o7.net. If *trackingServer* is not defined (and *dc* is not defined), data goes to 112.2o7.net.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	""

A list of Adobe data centers can be found [here](#).

trackingServerSecure

The *trackingServerSecure* variable is used for first-party cookie implementation to specify the domain at which the image request and cookie is written.

Used for secure pages. If *trackingServerSecure* is not defined, SSL data goes to *trackingServer*.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	N/A	N/A	""

transactionID

Integration Data Sources use a **transaction ID** to tie offline data to an online transaction (like a lead or purchase generated online).

Each unique *transactionID* sent to Adobe is recorded in preparation for a **Data Sources** upload of offline information about that transaction. See [Data Sources](#).

Max Size	Debugger Parameter	Reports Populated	Default Value
100 bytes	xact	n/a	""

Enabling Transaction ID Storage

Before *transactionID* values are recorded, **Transaction ID Storage** must be enabled for the report suite selected in the Report Suite Manager. This setting is located at

Analytics > Admin > Report Suites > Edit Settings > General > General Account Settings.

To see whether *transactionID Storage* is enabled for a report suite, go to

Analytics > Admin > Data Sources > Manage

Syntax and Possible Values

```
s.transactionID="unique_id"
```

The *transactionID* should contain only alphanumeric characters. If multiple **transactionIDs** should be recorded in a single hit, you can use a comma to delimit multiple values.

Examples

```
s.transactionID="11123456"
```

```
s.transactionID="lead_12345xyz"
```

```
s.transactionID=s.purchaseID
```

Pitfalls, Questions, and Tips

- If *transactionID* recording is not enabled, *transactionID* values will be discarded and unavailable for use with **Integration Data Sources**. Make sure to set a conversion variable or event (an eVar or the events variable) on the page where *transactionID* is set. Otherwise, no data is recorded for the *transactionID*.
- If you are recording **transactionIDs** for multiple systems, such as purchases and leads, make sure the value in *transactionID* is always unique. This can be accomplished by adding a prefix to the ID, such as lead_1234 and purchase_1234. **Integration Data Sources** do not function as expected (**Data Source** data will tie to the wrong data) if a unique *transactionID* is seen twice.
- By default, *transactionID* values are remembered for 90 days. If your offline interaction process is longer than 90 days, contact Customer Care to have the limit extended.



Note: The *transactionID* variable can contain any character other than a comma. It should be in the same location where the character limit (100 bytes) is specified. If multi-byte characters are used, multi-byte character support must be enabled in order to avoid problems with unexpected characters in the *transactionID*.

visitorID

Visitors can be identified by the *visitorID* variable or by IP address/User Agent.

The *visitorID* can be up to 100 alpha-numeric characters and must not contain a hyphen.


If you explicitly set a custom ID, it will always be used before the other ID methods.

This is the order of use: s.visitorID > s_vi > s_fid > IP/UA.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 bytes	vid	n/a	""

Syntax and Possible Values

```
s.visitorID="visitor_id"
```

 **Note:** The `visitorID` variable should not contain a hyphen.

Examples

```
s.visitorID="abc123"
```

Configuration Settings

None

visitorNamespace

The `visitorNamespace` variable is used to identify the domain with which cookies are set.

If `visitorNamespace` is used in your JavaScript file, do not delete or alter it. If `visitorNamespace` changes, all visitors reported in Analytics may become new visitors. Visitor history becomes disconnected from current and future traffic. Do not alter this variable without approval from an Adobe representative.

Max Size	Debugger Parameter	Reports Populated	Default Value
N/A	ns	N/A	""

Analytics uses a cookie to uniquely identify visitors to your site. If `visitorNamespace` is not used, the cookie is associated 2o7.net. If `visitorNamespace` is used, the cookie is associated with a sub-domain of 2o7.net. All visitors to your site should have their cookies associated with the same domain or sub-domain.

The reason for using the `visitorNamespace` variable is to avoid the possibility of overloading a browser's cookie limit. Internet Explorer imposes a limit of 20 cookies per domain. By using the `visitorNamespace` variable, other companies' Analytics cookies will not conflict with your visitors' cookies.

Syntax and Possible Values

The value of `visitorNamespace` must be provided by Adobe and is a string of ASCII characters that don't contain commas, periods, spaces, or special characters.

```
s.visitorNamespace="company_specific_value"
```

Visitor Identification across Report Suites

If you do not specify a `visitorNamespace`, each report suite in your company receives its own visitor ID cookie written as `s_vi_[random string]`. If you specify `visitorNamespace`, the same `s_vi` cookie will be used for all report suites that send data to the specified `trackingServer`. If you have implemented multi-suite tagging, make sure you specify the visitor namespace so the same cookie is used by each report suite.

Examples

```
s.visitorNamespace="company_name"
```

```
s.visitorNamespace="Adobe"
```

Configuration Settings

None

zip

The `state` and `zip` variables are conversion variables.

They are like eVars in that they capture events, but unlike eVars, they don't persist. The *zip* and *state* variables are like eVars that expire immediately.

Max Size	Debugger Parameter	Reports Populated	Default Value
50 bytes	zip	Conversion > Visitor Profile > ZIP/Postal Codes	""

Since the *state* and *zip* variables expire immediately, the only events associated with them are events fired on the same page that are populated. For example, if you are using *zip* to compare conversion rates by Zip Code, you should populate *zip* on every page of the checkout process. Adobe recommends using the billing address as the source for the Zip Code. You may choose to use the shipping address instead (assuming there is only one shipping address for the order). A media site may choose to use *zip* and *state* for registration or ad click-through tracking.

Syntax and Possible Values

```
s.zip="zip_code"
```

The *zip* variable does not impose any value or format restrictions. There are no limitations on *zip* outside of the standard variable limitations.

Examples

```
s.zip="92806"
```

```
s.zip="92806-4115"
```

Configuration Settings

None

Pitfalls, Questions, and Tips

- Populate **zip** on every page in which a relevant event is fired (such as each page of the checkout process).
- The *zip* and *state* variables act like eVars that expire on the Page View.

Additional eVars and Events

If you want to track additional information, but don't have enough variables to do so, you now have access to additional eVars and success events:



Note: JavaScript H-Code does not support these additional eVars and events.

Entitlements to Custom Dimensions and Events

Adobe Analytics Product			
Adobe Analytics - Foundation	75 props	200 eVars	1000 Events
Adobe Analytics - <i>Select</i>	75 props	200 eVars	1000 Events
Adobe Analytics - <i>Prime</i>	75 props	200 eVars	1000 Events
Adobe Analytics - <i>Ultimate</i>	75 props	250 eVars	1000 Events

Populating Variables and Events

- Variables can be populated in the data collection library if you are using these versions of AppMeasurement:
 - AppMeasurement for JavaScript version 1.4+
 - AppMeasurement for Flash version 3.9+
 - AppMeasurement for Java version 1.2.8+
 - AppMeasurement for Silverlight/.NET version 1.4.2+
 - AppMeasurement for PHP version 1.2.2+
- If you are on an earlier version of code, or on JavaScript H.*, you can populate context data and use processing rules to populate variables.
- All versions of data collection code can populate events 101-1000.
- Processing rules can be used to populate eVars 76-250 based on context data or other variables.

Frequently Asked Questions

• Will all Adobe Analytics interfaces have immediate access to these new variables?

These interfaces will have immediate access: Analysis Workspace, Reports & Analytics, Report Builder, Ad Hoc Analysis, APIs, and Data Workbench.

• Will these additional eVars and events automatically show up in my data feeds?

Data feeds will have access to the new variables and events once enabled. New eVar columns will not appear until you choose to include them. However, new events will appear in the event_list column as soon as they're enabled, and the events lookup table contains the event names for those event IDs. Do not enable new events unless you are ready to consume them in Data Feeds.

• How do I request new data feed columns?

To request new columns, refer to [Configuring Data Feeds](#).

• What if I am an Analytics Ultimate customer who wants to go back to Analytics Prime and I currently have more than 200 eVars enabled?

Adobe will not disable any of your existing eVars, but you will not be able to enable more. If you disable eVars, you cannot turn them back on until you are below the Analytics Prime limit of 200 enabled eVars.

Variables and Limitations

High-level look at variables and their limitations.



Note: See [Configuration Variables](#) and [Page Variables](#) for an in-depth look at the most common Analytics variables.

The following table provides at-a-glance information about Analytics variables:

Variable	Description
s_account	Determines the report suite where data is stored and reported.
browserHeight	Displays the height of the browser window.
browserWidth	Displays the width of the browser window.
campaign	Identifies marketing campaigns used to bring visitors to your site. The value of <i>campaign</i> is usually taken from a query string parameter.

Variable	Description
channel	Usually identifies a section of your website. For example, a merchant may have sections such as Electronics, Toys, or Apparel. A media site may have sections such as News, Sports, or Business.
charSet	Translates the character set of the Web page into UTF-8.
colorDepth	Displays the number of bits used to display color on each pixel of the screen.
connectionType	Indicates (in Microsoft Internet Explorer) whether the browser is configured on a LAN or modem connection.
cookieDomainPeriods	Determines the domain on which the Analytics visitor ID (<i>s_vi</i>) cookie will be set by determining the number of periods in the domain of the page URL. For <i>www.mysite.com</i> , <i>cookieDomainPeriods</i> should be "2." For <i>www.mysite.co.jp</i> , <i>cookieDomainPeriods</i> should be "3."
cookieLifetime	Used by both JavaScript and Analytics servers to determine the lifespan of a cookie.
cookiesEnabled	Indicates whether a first-party session cookie could be set by JavaScript.
currencyCode	Determines the conversion rate to be applied to revenue as it enters the Analytics databases. Analytics databases store all monetary amounts in a currency of your choice. If that currency is the same as that specified in <i>currencyCode</i> , or <i>currencyCode</i> is empty, no conversion is applied.
dc	Lets you set the data center to which your data is sent.
doPlugins	<i>doPlugins</i> is a reference to the <i>s_doPlugins</i> function. It allows the <i>s_doPlugins</i> function to be called at the appropriate location within the JavaScript file.
dynamicAccountList	Dynamically selects a report suite to which data is sent. The <i>dynamicAccountList</i> variable contains the rules that used to determine the destination report suite.
dynamicAccountMatch	Uses the DOM object to retrieve the section of the URL to which all rules in <i>dynamicAccountList</i> are applied. This variable is only valid when <i>dynamicAccountSelection</i> is set to 'True.'
dynamicAccountSelection	Lets you dynamically select the report suite based on the URL of each page.
dynamicVariablePrefix	Allows deployment to flag variables that should be populated dynamically. Cookies, request headers, and image query string parameters are available to be populated dynamically.
eVarN	Used for building custom reports within the Analytics Conversion Module . When an eVar is set to a value for a visitor, Analytics remembers that value until it expires. Any success events that a visitor encounters while the eVar value is active are counted toward the eVar value.
events	Records common shopping cart success events and custom success events.
fpCookieDomainPeriods	Determines the domain on which Analytics cookies other than the visitor ID (<i>s_vi</i>) cookie will be set by determining the number of periods in the domain of the page.
hierN	Determines the location of a page in your site's hierarchy. This variable is most useful for sites that have more than three levels in the site structure.
homepage	Indicates (in Internet Explorer) whether the current page is set as the user's home page.
javaEnabled	Indicates whether Java is enabled in the browser.

Variable	Description
javascriptVersion	Displays the version of JavaScript supported by the browser.
linkDownloadFileTypes	A comma-separated list of file extensions. If your site contains links to files with any of these extensions, the URLs of these links appear in the File Downloads report.
linkExternalFilters	If your site contains many links to external sites, and you don't want to track all exit links, <i>linkExternalFilters</i> can be used to report on a specific subset of exit links.
linkInternalFilters	Determines which links on your site are exit links. It is a comma-separated list of filters that represent the links that are part of the site.
linkLeaveQueryString	Determines whether or not the query string should be included in the Exit Links and File Download reports.
linkName	An optional variable used in Link Tracking that determines the name of a custom, download, or exit link. The <i>linkName</i> variable is not normally needed because the third parameter in the <i>tl()</i> function replaces it.
linkTrackEvents	Contains the events that should be sent with custom, download, and exit links. This variable is only considered if <i>linkTrackVars</i> contains "events."
linkTrackVars	A comma-separated list of variables that will be sent with custom, exit, and download links. If <i>linkTrackVars</i> is set to "", all variables that have values are sent with link data.
linkType	An optional variable used in link tracking that determines which report a Link Name or URL will appear (custom, download, or exit Links). <i>linkType</i> is not normally needed because the second parameter in the <i>tl()</i> function replaces it.
mediaLength	Specifies the total length of the media being played.
mediaName	Specifies the name of the video or media item. It is only available via the Data Insertion API and Full Processing Data Source .
mediaPlayer	Specifies the player used to consume a video or media item.
mediaSession	Specifies the segments of a video or media asset consumed.
Media.trackEvents	Identifies which events should be sent with a media hit. It is only applicable with JavaScript ActionSource ..
Media.trackVars	Identifies which variables should be sent with a media hit. It is only applicable with JavaScript ActionSource ..
mobile	Controls the order in which cookies and subscriber ids are used to identify visitors.
s_objectID	A global variable that should be set in the onClick event of a link. By creating a unique object ID for a link or link location on a page, you can improve visitor click map tracking or use visitor click map to report on a link type or location, rather than the link URL.
pageName	Contains the name of each page on your site. If <i>pageName</i> is blank, the URL is used to represent the page name in Analytics.
pageType	Used only to designate a 404 Page Not Found Error page. It only has one possible value, which is "errorPage." On a 404 Error page, the <i>pageName</i> variable should not be populated.

Variable	Description
pageURL	In rare cases, the URL of the page is not the URL that you would like reported in Analytics. To accommodate these situations, Analytics offers the <i>pageURL</i> variable, which overrides the actual URL of the page.
plugins	On Netscape and Mozilla-based browsers, lists the plugins installed in the browser.
products	Used for tracking products and product categories as well as purchase quantity and purchase price. The <i>products</i> variable should always be set in conjunction with a success event. Optionally, the <i>products</i> variable can track custom numeric and currency events, as well as Merchandising eVars.
propN	Used for building custom reports within the Analytics Traffic Module . props may be used as counters (to count the number of times a page view is sent), for pathing reports, or in correlation reports.
purchaseID	Used to keep an order from being counted multiple times by Analytics. Whenever the purchase event is used on your site, you should use the <i>purchaseID</i> variable.
referrer	Restores lost referrer information.
resolution	Displays the monitor resolution of the visitor viewing the Web page.
server	Shows either the domain of a Web page (to show which domains people come to) or the server serving the page (for a load balancing quick reference).
state	Captures the state in which a visitor to your site resides.
trackDownloadLinks	Set <i>trackDownloadLinks</i> to 'true' if you want to track links to downloadable files on your site. If <i>trackDownloadLinks</i> is 'true,' <i>linkDownloadFileTypes</i> determines which links are downloadable files.
trackExternalLinks	If <i>trackExternalLinks</i> is 'true,' <i>linkInternalFilters</i> and <i>linkExternalFilters</i> determines whether any link clicked is an exit link.
trackingServer	Used for first-party cookie implementation to specify the domain at which the image request and cookie is written. Used for non-secure pages.
trackingServerSecure	Used for first-party cookie implementation to specify the domain at which the image request and cookie is written. Used for secure pages.
trackInlineStats	Determines whether visitor click map data is gathered.
transactionID	Ties offline data to an online transaction (like a lead or purchase generated online). Each unique <i>transactionID</i> sent to Adobe is recorded in preparation for a Data Sources upload of offline information about that transaction. See the Data Sources Guide .
s_usePlugins	If the <i>s_doPlugins</i> function is available and contains useful code, s_usePlugins should be set to 'true.' When usePlugins is 'true,' the <i>s_doPlugins</i> function is called prior to each image request.
visitorID	Visitors may be identified by the <i>visitorID</i> tag, or by IP address/User Agent. The <i>visitorID</i> may be up to 100 alphanumeric characters and must not contain a hyphen.
visitorNamespace	If <i>visitorNamespace</i> is used in your JavaScript file, do not delete or alter it. This variable is used to identify the domain with which cookies are set. If <i>visitorNamespace</i> changes, all visitors reported in Analytics may become new visitors. In short, do not alter this variable without approval from an Adobe consultant.

Variable	Description
zip	Captures the ZIP code in which a visitor to your site resides.

Illegal JavaScript Characters

Characters and strings that are never allowed in JavaScript variables.

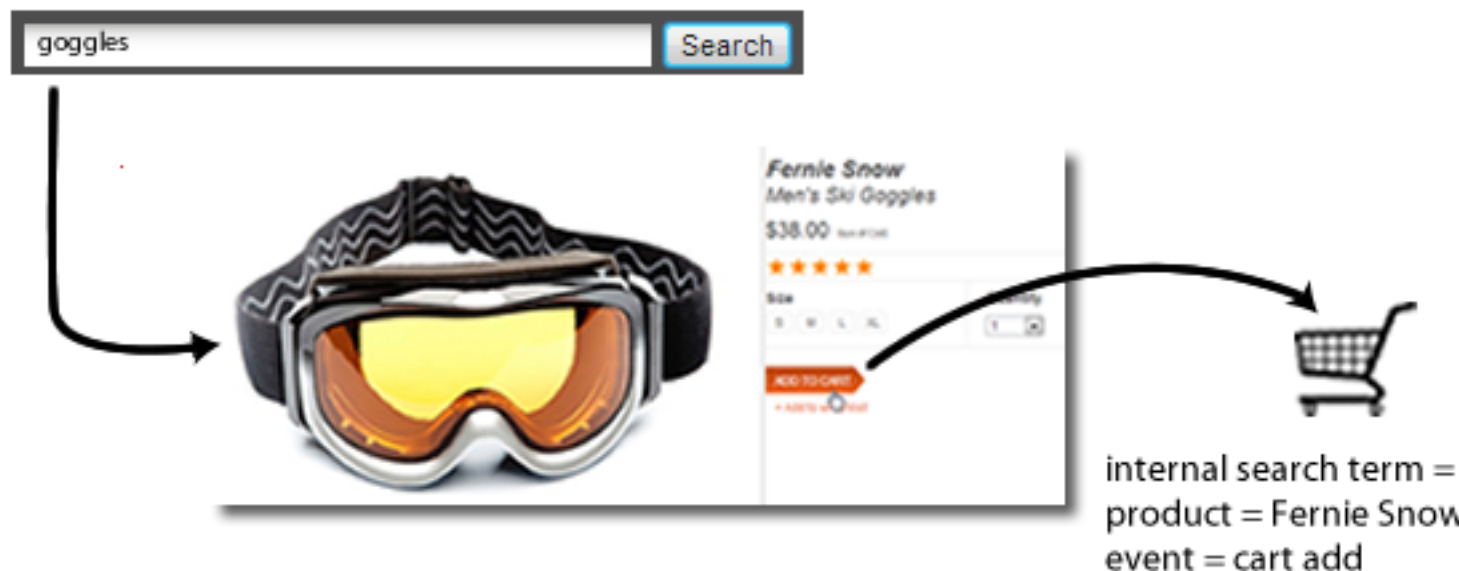
- Tab (0x09)
- Carriage return (0x0D)
- Newline (0x0A)
- ASCII characters with codes above 127 (unless multi-byte characters are enabled and *charSet* is populated appropriately)
- HTML tags (e.g. `` or `™`)

Many variables, most notably products, hierarchy, and events, have additional limitations or syntax requirements. For individual variable limitations and syntax requirements, see the section corresponding to the *s_account* variable parameters.

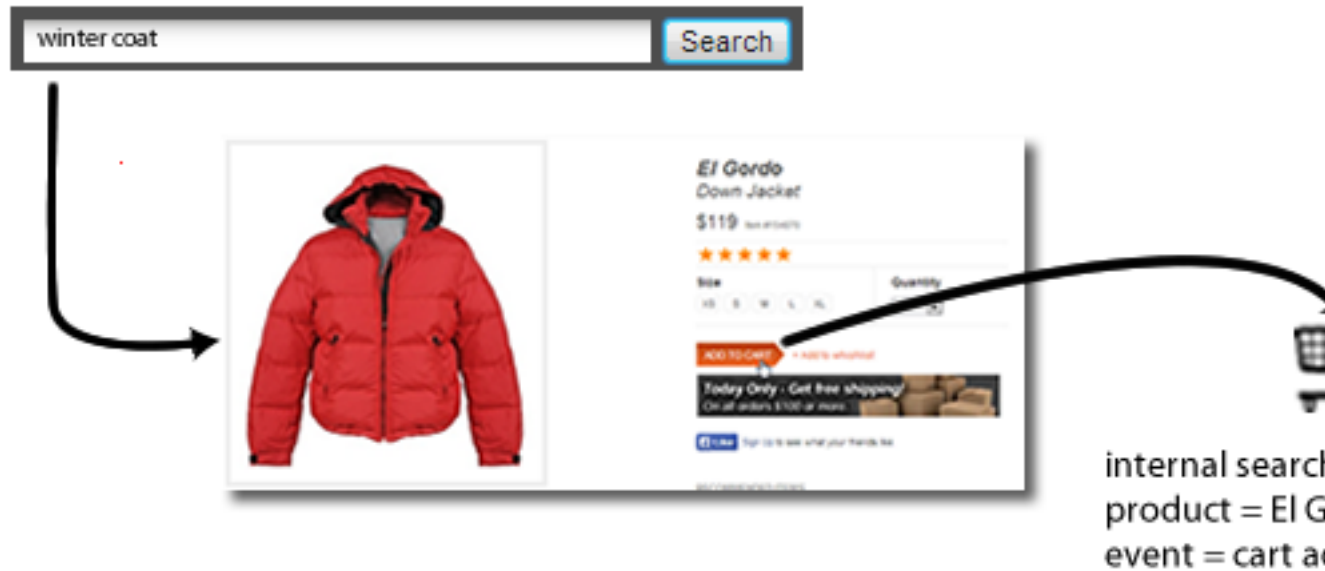
Merchandising Variables

When measuring the success of external campaigns or external search terms, you typically want a single value to receive credit for any success events that occur. For example, if a customer clicks a link in an email campaign to visit your website, all purchases made as a result should be credited to that campaign.

But what about events that are driven by internal search or by category browsing when a customer is looking for multiple items? For example, a customer searches your site for "goggles" and then adds a pair to the cart:



Before checkout, the customer searches for "winter coat", and then adds a down jacket to the to the cart:



When this purchase is completed, assuming the allocation wasn't changed from Most Recent, you'll have an internal search for "winter coat" credited with the purchase of a pair of goggles. Good for "winter coat", but bad for marketing decisions:

Internal Search Term	Revenue
winter coat	\$157

How merchandising variables solve this problem

Cross-category merchandising variables, or "merchandising eVars", let you assign the current value of an eVar to a product at the time a success event takes place. This value remains tied to that product, even if one or more new values are later set for that particular eVar.

If merchandising is enabled for the eVar in the previous example, the search term "goggles" is tied to the Fernie Snow Goggles, and the search term "winter coat" is tied to the El Gordo Down Jacket. Merchandising variables allocate revenue at the product level, so each term receives credit for the amount of revenue for the product to which the term was associated:

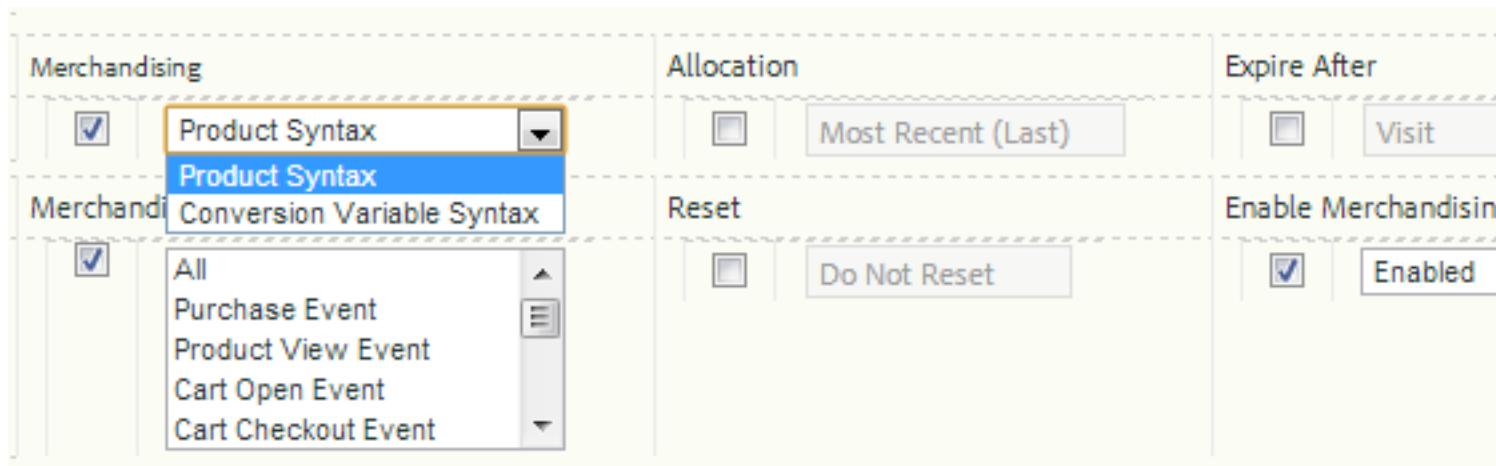
Internal Search Term	Revenue
winter coat	\$119
goggles	\$38


Implementing a Merchandising Variable

Describes how to enable and implement a merchandising variable.

Enable a Merchandising Variable

Merchandising can be enabled for any custom eVar on the **Admin Tools > Report Suites > Conversion Variables** page (you no longer need to call Adobe):



Setting	Description
Expire After	Determines how long merchandising values should persist.
Merchandising	<p><i>Product Syntax</i>: The value is set within <code>s.products</code>.</p> <p><i>Conversion Variable Syntax</i>: The value is set in the designated merchandising <code>s.eVar</code>.</p>
Merchandising Binding Event (Conversion variable syntax only)	<p>Indicates when a product should be tied to the current merchandising category. Multiple events may be selected by holding down Ctrl and clicking on multiple items in the list.</p> <p> Note: When "Product Syntax" is selected, you can not choose an event (it is disabled, but not grayed out). You can select an event only when the "Conversion Variable Syntax" is selected.</p>

Implementing Using Product Syntax

When Product Syntax is enabled, the merchandising category is populated directly within the products variable, so selecting and setting a binding event is not required. This is the recommended method and should be used unless the value is not available to set in `s.products` when the success event takes place.

• Syntax

```
s.products="category;product;quantity;price;event_incrementer;eVarN=merch_category|eVarM=merch_category2"
```

• Example

```
s.events="prodView"
s.products=";Fernie Snow Goggles;;;eVar1=goggles"
In
```

The value "goggles" for eVar1 is assigned to the product "Fernie Snow Goggles". All subsequent success events (product adds, checkouts, purchases, and so on) that involve this product are credited to "goggles".

Implementing Using Conversion Variable Syntax

Conversion Variable Syntax should be used when the eVar value is not available to set in `s.products`. This typically means that your page has no context of the merchandising channel or finding method. In these cases you must set the merchandising variable before you arrive at the product page, and the value persists until the binding event occurs.

When the binding event selected during configuration occurs, the persisted value of the eVar is associated with the product. For example, if `prodView` is specified as the binding event, the merchandising category is tied to the current product list only at the time the event occurs. Only subsequent binding events can update a merchandising eVar that has already been assigned to a product.

• Syntax

On the same or previous page before the binding event:

```
s.eVar1="merchandising_category"
```

On the page where the binding event occurs:

```
s.events="prodView"  
s.products="category;product "
```

• Example

Page 1 of the visit:

```
s.eVar1="Outdoors:Ski Goggles"
```

Page 2 of the visit:

```
s.events="prodView"  
s.products=";Fernie Snow Goggles"
```

The value "Outdoors:Ski Goggles" for eVar1 is assigned to the product "Fernie Snow Goggles". All subsequent success events (product adds, checkouts, purchases, and so on) that involve this product are credited to "goggles".

Additionally, the current value of the merchandising variable is tied to all subsequent products until one of the following conditions is met:

- eVar expires (based on the "Expire After" setting)
- The merchandising eVar is overwritten with a new value.

For more information, see "[Advanced Conversion Syntax Merchandising](#)" at analyticstodemystified.com.

Instances on Merchandising Variables

Describes how instances are counted on merchandising variables.

Instances are not currently supported for merchandising variables. If you notice instances in a report containing a merchandising variable, it indicates that the eVar is being set in some locations outside of the products string and should not be considered as a true count of instances for the selected merchandising variable.

If you are using Conversion Variable Syntax, an instance is counted each time the variable is set. However, the instance is attributed to "None" unless the following occurs each time the variable is set:

- A binding event is set.
- The products variable is set.

- The merchandising eVar has a value.

For example, the following instance of eVar1 is allocated to "Outdoors:Ski Goggles":

```
s.eVar1="Outdoors:Ski Goggles"
s.events="prodView"
s.products=";Fernie Snow Goggles"
```

However, in the next example, the instance of eVar1 is allocated to "None" since all conditions are not met when the eVar is set (there is no binding event and the products variable is not set):

Page 1 of the visit:

```
s.eVar1="Outdoors:Ski Goggles"
```

Page 2 of the visit:

```
s.events="prodView"
s.products=";Fernie Snow Goggles"
```

Allocation to "None" occurs if you set a value for an eVar on a page where no binding event occurs, or if you set the eVar value in the products string without a binding event.



Note: The current functionality for counting instances on merchandising variables is being reviewed and is scheduled to change in an upcoming release.

The s.t() Function - Page Tracking

The **s.t()** function is what compiles all the variables defined on that page into an image request and sends it to our servers.

Properties of the Function

- Removing the **s.t()** call prevents any data from reaching Analytics. Multiple **s.t()** calls fires multiple image requests (doubling the reported traffic on your site).
- If you wish to fire more than one image request on a single page load, using the **s.tl()** function is recommended.
- Triggering this function always increases **pageviews** and always include the **s.pageName** variable.

Implementation

Upon generating code within the **code manager**, you are given the following at the bottom of the page code:

```
var s_code=s.t();if(s_code)document.write(s_code)//--></script>
<script language="JavaScript"
type="text/javascript"><!--if(navigator.appVersion.indexOf('MSIE')>=0)document.write(unescape('%3C')+'!'+'-')//--></script>
<noscript></noscript>
```

Each line of code has a specific purpose:

```
var s_code=s.t();if(s_code)document.write(s_code)//-->
```

This line of code is what actually fires the Javascript function. The **s_code** variable and it's accompanying `document.write` method is for browsers that don't support image objects (Netscape browsers prior to version 3 and Internet Explorer prior to version 4; estimated less than .5% of all internet users).

```
<script language="JavaScript"
type="text/javascript"><!--if(navigator.appVersion.indexOf('MSIE')>=0)document.write(unescape('%3C')+'!'+'-')//--></script>
<noscript>
```


For any additional questions about the **s.t()** function, contact your organization's Account Manager. They can arrange a meeting with an Adobe Implementation Consultant, who can provide assistance.

The s.tl() Function - Link Tracking

File downloads and exit links can be automatically tracked based on parameters set in the AppMeasurement for JavaScript file.

If needed, these types of links can be manually tracked using custom link code as explained below. In addition, custom link code can be used to track generic custom links that can be used for a variety of tracking and reporting needs.

s.tl() Parameter Reference

Variable	Description								
this	<p>The first argument should always be set either to this (default) or true. The argument refers to the object being clicked; when set to "this," it refers to the HREF property of the link.</p> <p>If you are implementing link tracking on an object that has no HREF property, you should always set this argument to "true."</p> <p>Because clicking a link often takes a visitor off the current page, a 500ms delay is used to ensure that an image request is sent to Adobe before the user leaves the page. This delay is only necessary when leaving the page, but is typically present when the s.tl() function is called. If you want to disable the delay, pass the keyword 'true' as the first parameter when calling the s.tl() function.</p>								
linkType	<p><code>s.tl(this,linkType,linkName, variableOverrides, doneAction)</code></p> <p>linkType has three possible values, depending on the type of link that you want to capture. If the link is not a download or an exit link, you should choose the Custom links option.</p> <table border="1"> <thead> <tr> <th>Type</th> <th>linkType value</th> </tr> </thead> <tbody> <tr> <td>File Downloads</td> <td>'d'</td> </tr> <tr> <td>Exit Links</td> <td>'e'</td> </tr> <tr> <td>Custom Links</td> <td>'o'</td> </tr> </tbody> </table>	Type	linkType value	File Downloads	'd'	Exit Links	'e'	Custom Links	'o'
Type	linkType value								
File Downloads	'd'								
Exit Links	'e'								
Custom Links	'o'								
linkName	This can be any custom value, up to 100 characters. This determines how the link is displayed in the appropriate report.								
variableOverrides	(Optional, pass null if not using) This lets you change variable values for this single call, It is similar to other AppMeasurement libraries.								
useForcedLinkTracking	<p>This flag is used to disable forced link tracking for some browsers. Forced link tracking is enabled by default for FireFox 20+ and WebKit browsers.</p> <p>Default Value</p> <p>true</p>								

Variable	Description
	<p>Example</p> <pre>s.useForcedLinkTracking = false</pre>
forcedLinkTrackingTimeout	<p>The maximum number of milliseconds to wait for tracking to finish before performing the doneAction that was passed into <code>s.tl</code>. This value specifies the maximum wait time. If the track link call completes before this timeout, the doneAction is executed immediately. If you notice that track link calls are not completing, you might need to increase this timeout.</p> <p>Default Value</p> <p>250</p> <p>Example</p> <pre>s.forcedLinkTrackingTimeout = 500</pre>
doneAction	<p>An optional parameter to specify a navigation action to execute after the track link call completes when <code>useForcedLinkTracking</code> is enabled.</p> <p>Syntax</p> <pre>s.tl(linkObject,linkType,linkName,variableOverrides,doneAction)</pre> <p>doneAction : (optional) Specifies the action to take after the link track call is sent or has timed out, based on the value specified by:</p> <pre>s.forcedLinkTrackingTimeout</pre> <p>The doneAction variable can be the string <code>navigate</code>, which causes the method to set <code>document.location</code> to the href attribute of linkObject. The doneAction variable can also be a function allowing for advanced customization.</p> <p>If providing a value for doneAction in an anchor onClick event, you must return <code>false</code> after the <code>s.tl</code> call to prevent the default browser navigation.</p> <p>To mirror the default behavior and follow the URL specified by the href attribute, provide a string of <code>navigate</code> as the doneAction.</p> <p>Optionally, you can provide your own function to handle the navigation event by passing this function as the doneAction.</p> <p>Examples</p> <pre>Click Here Click Here</pre>

Example

The following example of an **s.tl()** function call uses the default 500 ms delay to ensure data is collected before leaving the page.

```
s.tl(this,'o','link name');
```

The following example disables the 500 ms delay, if the user is not going to leave the page, or whenever the object being clicked has no HREF.


```
s.tl(true,'o','link name');
```

The 500ms delay is a maximum delay. If the image requested returns in less than 500 ms, the delay stops immediately. This allows the visitor to move onto the next page or next action within the page.

The following examples are for handling custom links on WebKit browsers:


```
<a href="..." onclick="s.tl(this,'o','MyLink',null,'navigate');return false">Click Here</a>
```


```
<a href="#" onclick="s.tl(this,'o','MyLink',null,
function(){if(confirm('Proceed?'))document.location=...});return false">Click Here</a>
```

 **Note:** Uses of custom link code are often very specific to your Web site and reporting needs. You can contact your Adobe Consultant or Customer Care before implementing custom link code to understand the possibilities available to you and how best to leverage this feature based on your business needs.

The basic code to track a link using custom link code is shown in the following example:

```
<a href="index.html" onClick="s.tl(this,'o','Link Name')">My Page</a>
```

 **Note:** The `s_gi` function must contain your report suite ID as a function parameter. Be sure to swap out `rsid` for your unique report suite ID.

 **Note:** If the link name parameter is not defined, the URL of the link (determined from the "this" object) is used as the link name.

Analytics variables can be defined as part of custom link code.

Automatic Tracking of Exit Links and File Downloads

The JavaScript file can be configured to automatically track file downloads and exit links based on parameters that define file download file types and exit links.

The parameters that control automatic tracking are as follows:

```
s.trackDownloadLinks=true
s.trackExternalLinks=true
s.linkDownloadFileTypes="exe,zip,wav,mp3,mov,mpg,avi,doc,pdf,xls"
s.linkInternalFilters="javascript:,mysite.com,[more filters here]"
s.linkLeaveQueryString=false
```

The parameters `trackDownloadLinks` and `trackExternalLinks` determine if automatic file download and exit link tracking are enabled. When enabled, any link with a file type matching one of the values in `linkDownloadFileTypes` is automatically tracked as a file download. Any link with a URL that does not contain one of the values in `linkInternalFilters` is automatically tracked as an exit link.

In JavaScript H.25.4 (released February 2013), automatic exit link tracking was updated to always ignore links with HREF attributes that start with `#`, `about:`, or `javascript:`.

Example 1

The file types `jpg` and `aspx` are not included in `linkDownloadFileTypes` above, therefore no clicks on them are automatically tracked and reported as file downloads.

The parameter `linkLeaveQueryString` modifies the logic used to determine exit links. When `linkLeaveQueryString=false`, exit links are determined using only the domain, path, and file portion of the link URL. When `linkLeaveQueryString=true`, the query string portion of the link URL is also used to determine an exit link.

Example 2

With the following settings, the example below will be counted as an exit link:

```
//JS file
s.linkInternalFilters="javascript:,mysite.com"
s.linkLeaveQueryString=false

//HTML file
<a href='http://othersite.com/index.html?r=mysite.com'>Visit Other Site!</a>
```

Example 3

With the following settings, the link below is not counted as an exit link:

```
//JS file
s.linkInternalFilters="javascript:,mysite.com"
s.linkLeaveQueryString=true

//HTML
<a href='http://othersite.com/index.html?r=mysite.com'>Visit Other Site</a>
```



Note: A single link can be tracked only as a file download or exit link, with file download taking priority. If a link is both an exit link and file download based on the parameters `linkDownloadFileTypes` and `linkInternalFilters`, it is tracked and reported as a file download and not an exit link. The following table summarizes the automatic tracking of file downloads and exit links.

Manual Link Tracking Using Custom Link Code

Custom link code lets file downloads, exit links, and custom links be tracked in situations where automatic tracking is not sufficient or applicable.

Custom link code is typically implemented by adding an **onClick** event handler to a link or by adding code to an existing routine. It can be implemented from essentially any JavaScript event handler or function.

Link Tracking consists of calling the `AppMeasurement` for JavaScript function whenever the user performs actions that generate data you want to capture. This function, `s.tl()`, is either called directly in an event handler, such as **onClick** or **onChange**, or from within a separate function. This `s.tl()` function has five arguments. The first three are required:

```
s.tl(this,linkType,linkName, variableOverrides, doneAction)
```

Custom Link Tracking on FireFox and WebKit Browsers

JavaScript H.25 includes an update to ensure that link tracking completes successfully on WebKit browsers (Safari and Chrome). JavaScript H.26 includes an update to ensure that link tracking completes successfully on FireFox 20+.

After this update, download and exit links that are automatically tracked (determined by `s.trackDownloadLinks` and `s.trackExternalLinks`) are tracked successfully. If you are tracking custom links using manual JavaScript calls, you need to modify how these calls are made. For example, exit and download links are often tracked using code similar to the following:

```
<a href="http://anothersite.com" onclick="s.tl(this,'e','AnotherSite',null)">
```

Internet Explorer executes the track link call and open the new page. Other browsers might cancel execution of the track link call when the new page opens. This often prevents track link calls from completing.

To work around this behavior, H.25 (released July 2012) includes an overloaded track link method (`s.tl`) that forces browsers with this behavior to wait for the track link call to complete. This new method executes the track link call and handles the navigation event, instead of using the default browser action. This overloaded method requires an additional parameter, called **doneAction**, to specify the action to take when the link tracking call completes.

To use this new method, update calls to `s.tl` with an additional **doneAction** parameter, similar to the following:

```
<a href="http://anothersite.com"
onclick="s.tl(this,'e','AnotherSite',null,'navigate');return false">
```

Passing `navigate` as the **doneAction** mirrors the default browser behavior and opens the URL specified by the `href` attribute when the tracking call completes.

In JavaScript H.25.4 (released February 2013), the following scope limitations were added to links tracked when `useForcedLinkTracking` is enabled. The automatic forced link tracking applies only to:

- `<A>` and `<AREA>` tags.
- The tag must have an `HREF` attribute.
- The `HREF` can't start with `#`, `about:`, or `javascript:`.
- The `TARGET` attribute must not be set, or the `TARGET` needs to refer to the current window (`_self`, `_top`, or the value of `window.name`).

Link Tracking Using an Image Request

Links can be tracked without calling the `s.tl()` function by constructing an image request.

Image requests are hard coded by adding the "pe" parameter to your image request `src` parameter as follows:

```
pe=[type]
```

Where `[type]` is replaced with the type of link you want to track:

- `Ink_d` = download
- `Ink_e` = exit
- `Ink_o` = custom

Additionally, a link URL can be specified by passing the URL in the `pev1` parameter:

```
pev1=mylink.com
```

A link name can be specified by passing the name in the `pev2` parameter:

```
pev2=My%20Link
```

For example:

```

```

Setting Additional Variables for File Downloads, Exit Links, and Custom Links

Two parameters (`linkTrackVars` and `linkTrackEvents`) control which Analytics variables are set for file downloads, exit links, and custom links.

They are, by default, set within the JS file as follows:

```
s.linkTrackVars="None"
```

```
s.linkTrackEvents="None"
```

The `linkTrackVars` parameter should include each variable that you want to track with every file download, exit link, and custom link. The `linkTrackEvents` parameter should include each event you want to track with every file download, exit link, and custom link. When one of these link types occur, the current value of each variable identified is tracked.

For example to track `prop1`, `eVar1`, and `event1` with every file download, exit link, and custom link, use the following settings within the global JS file:

```
s.linkTrackVars="prop1,eVar1,events"
```

```
s.linkTrackEvents="event1"
```



Note: The variable `pageName` cannot be set for a file download, exit link, or custom link, because each of the link types is not a page view and does not have an associated page name.



Note: If `linkTrackVars` (or `linkTrackEvents`) is null (or an empty string), all Analytics variables (or events) that are defined for the current page are tracked. This most likely inflates instances of each variable inadvertently and should be avoided.

Best Practices

The settings for `linkTrackVars` and `linkTrackEvents` within the JS file affect every file download, exit link, and custom link. Instances of each variable and event can be inflated in situations where the variable (or event) applies to the current page, but not the specific file download, exit link, or custom link.

To ensure that the proper variables are set with custom link code, Adobe recommends setting `linkTrackVars` and `linkTrackEvents` within the custom link code, as follows:

```
<a href="index.html" onClick="
var s=s_gi('rsid');
s.linkTrackVars='prop1,prop2,eVar1,eVar2,events';
s.linkTrackEvents='event1';
s.prop1='Custom Property of Link';
s.events='event1';
s.tl(this,'o','Link Name');
">My Page
```

The values of `linkTrackVars` and `linkTrackEvents` override the settings in the JS file and ensure only the variables and events specified in the custom link code are set for the specific link.



Note: In the above example, the value for `prop1` is set within the custom link code itself. The value of `prop2` comes from the current value of the variable as set on the page.

Using Function Calls with Custom Link Code

Due to the complex nature of custom link code, you can consolidate the code into a self-contained JavaScript function (defined on the page or in a linked JavaScript file) and make calls to the function within the **onClick** handler.

For example, you could insert the following two functions in your `AppMeasurement.js` file, just below the `s.doPlugins()` function, and then use them throughout your site:

```
/* Set Click Interaction values (with timeout - H25 code and higher*/
function trackClickInteraction(name){
    var s=s_gi('rsid');
    s.linkTrackVars='prop42,prop35';
    s.prop42=name;
    s.prop35=s.pageName;
```

```

    s.tl(true,'o','track interaction',null,'navigate');
}
/* Set Click Interaction values (without timeout - pre H25 code*/
function trackClickInteraction(name){
    var s=s_gi('rsid');
    s.linkTrackVars='prop42,prop35';
    s.prop42=name;
    s.prop35=s.pageName;
    s.tl(true,'o','track interaction');
}

```



Note: If needed, you can pass the link type and link name as additional parameters for the JavaScript function.

You can use code similar to the following to call these functions:

```

<a href="http://www.your-site.com/some_page.php"
onclick="trackClickInteraction('this.href');">Link Text</a>

```

Avoiding Duplicate Link Counts

It is possible for the link to be double-counted in situations where the link is normally captured by automatic file download or exit link tracking.

For example, if you are tracking PDF downloads automatically, the following code results in a duplicate download count:

```

function trackDownload(obj) {
    var s=s_gi('rsid');
    s.linkTrackVars='None';
    s.linkTrackEvents='None';
    s.tl(obj,'d','PDF Document');
}

```

To ensure link double counting does not occur, use the following modified JavaScript function:

```

<script language=JavaScript>
function linkCode(obj) {
    var s=s_gi('rsid');
    s.linkTrackVars='None';
    s.linkTrackEvents='None';
    var lt=obj.href!=null?s.lt(obj.href):"";
    if (lt=="") { s.tl(obj,'d','PDF Document'); }
}

```

The last two lines of the code above modify the behavior of custom link code so only the automatic tracking behavior occurs, eliminating any possible double counting.

Popup Windows with useForcedLinkTracking

When `useForcedLinkTracking` is enabled, AppMeasurement overrides the default link behavior on some browsers to prevent the track link call from being canceled when the new page opens. AppMeasurement executes the track link call and handles the navigation event manually, instead of using the default browser action.

When tracking some links that open a popup window, the Chrome built-in popup blocker might prevent AppMeasurement from opening a popup window that would normally be allowed. To avoid this, add a `target="_blank"` attribute to calls that open a window:

```

<a href="./popup.html" target="_blank" onClick="<a href='./popup.html'
onclick="s.tl(this,'o','popup',null,'navigate');javascript:window.open('./popup.html','popup','width=550,height=450');
return false;">

```

This allows the link to be tracked and the popup to load as expected.

Links from URL Shorteners

Links from URL shortener services (such as bit.ly) are typically not tracked as page views or as referrers. These services return 301/302 redirects with the full URL to the web browser, which then sends a new, separate request to the full URL. The original referrer is preserved since the shortener is no longer in the loop, and there isn't an indication on the request that a redirect service was used to get the URL.

Due to the variety of services available, we recommend testing the specific scenarios in use on your site to determine the redirect mechanism used by the service.

Link Tracking Variables in doPlugins

To help manage link tracking, these following variables are populated before the `doPlugins` function runs.

Inside of `doPlugins`, you can use the following values to modify link tracking behavior. For example, you can abort tracking, or add additional variables to tracking requests.

Variable	Description	Impact
linkType	<p>Contains the automatically determined link type, if any. Can be set to one of the following:</p> <ul style="list-style-type: none"> • d (download) • e (exit) • o (custom/other) <p>This is the <code>pe</code> parameter in the image request.</p>	If set with <code>linkURL</code> or <code>linkName</code> , a server call is sent as a download, custom, or exit link.
linkName	<p>The name that will appear in the custom, download or exit link report. Truncated at 100 characters. Can be set to any string.</p> <p>This is the <code>pev2</code> parameter in the image request.</p>	If set with <code>linkType</code> , an image request will be sent as a download, custom or exit link
linkURL	<p>The URL of the link, which acts as the name if a <code>linkName</code> does not exist. Can be set to any URL string.</p> <p>This is the <code>pev1</code> parameter in the image request.</p>	If set with <code>linkType</code> , an image request will be sent as a download, custom or exit link
linkObject	The clicked object for reference. This is read-only.	No direct impact on measurement.

Example

```
function s_doPlugins(s) {
  if (s.linkType == "d" && s.linkURL.indexOf(".aspx?f=") {
    //special tracking for .aspx file download script
    s.eVar11 = s.linkURL.substring(s.linkURL.lastIndexOf("?f=") + 3, s.linkURL.length);
  }

  else if (s.linkType == "o" ) {
    // note: linkType is set to "o" only if you make a custom call
    // to s.tl() and set the link type to "o". Automatically tracked
    // links are set to "d" or "e" only.
    s.eVar10 = s.LinkURL;
  }
}
```


Validating File Downloads, Exit Links, and Custom Links

To fully validate download, exit, and custom links, Adobe recommends using a packet analyzer to examine the links in real-time.

File downloads, exit links, and custom links are not page views, so the **DigitalPulse Debugger** tool cannot be used to verify parameters and variable settings. You must use a [Packet Analyzers](#) to view track link data.

The s.sa() Function

The **s.sa() function** lets you dynamically change a report suite at any time on the page, before or after an image request fires.

If your organization wants to send data to different report suites without a page reload, using this function is strongly recommended.

This information is suited for advanced users who are well-versed in both reporting and implementation. Do not attempt to make any edits to your implementation without complete knowledge of its consequences. If you require implementation changes, contact your organization's Account Manager.

Properties of the Function

Setting this function takes all previously defined variables and lets them be used in a different report suite.

Implementation Examples

Sending video data to one report suite while sending the rest to another:

```
// Set in the core JS file by default
var s=s_gi('prodrsid');

// Define this when a user interacts with a video
s.sa('videorsid');
s.t(); // Sends an image request
```

Using s.sa() and multi-suite tagging:

```
// Set in the core JS file by default
var s=s_gi('rsid1');

// Call the function when you wish to change report suites
s.sa('rsid1,rsid2');
s.t();
```

The s.clearVars() Function

Clears the following values from the instance object. This function removes the elements (sets them as "undefined.")

- props
- eVars
- hier
- list
- events
- eventList
- products
- productsList

- channel
- purchaseID
- transactionID
- state
- zip
- campaign

For example:

```
s.clearVars()
```



Note: `clearVars()` is included in [About AppMeasurement for JavaScript](#) but is not available in H code and previous versions.

The `s_gi()` Function

The `s_gi()` function is used to create or find your instance of AppMeasurement by report suite ID. Internally, AppMeasurement keeps track of every instance created, and `s_gi()` returns the existing instance for a report suite if one exists. If an instance does not exist, a new instance is created and returned.

We recommend calling `s_gi()` before setting variables and making tracking calls throughout your page code. This ensures that the correct object is used to make the tracking call in the case that the `s` variable is inadvertently overwritten.

Using Multiple Report Suites

The object returned varies based on the report suite ID(s) passed. For example, if you make the following initial call to `s_gi()`:

```
var s=s_gi('rsid1,rsid2')
```

The following table outlines what is returned by subsequent calls:

Subsequent Call to <code>s_gi</code>	Description of Object Returned
<code>s=s_gi('rsid1,rsid2')</code>	The same object referenced earlier.
<code>s=s_gi('rsid1')</code>	A copy of the object created earlier, but not the original.
<code>s=s_gi('rsid1,rsid3')</code>	A copy of the object created earlier, but not the original.
<code>s=s_gi('rsid3')</code>	A new, empty object, with no config variables set (e.g. <code>linkTrackVars</code> is empty, as is <code>linkDownloadFileTypes</code>).

Util.cookieRead

Gets the value for a cookie.

Syntax:

```
s.Util.cookieRead(key)
```

Parameters:

Parameter	Description
key	(required) key to write value for in cookies.

Returns:

Cookie value or an empty string if the cookie is not found.

Example:

```
var myCookie = s.Util.cookieRead("my_cookie");
```

Util.cookieWrite

Writes a value to a cookie.

Syntax:

```
s.Util.cookieWrite(key, value [,expire])
```

Parameters:

Parameter	Description
key	(required) key to write value for in cookies.
value	(optional) value to write to cookies.
expire	(optional) Date object containing the expiration date for the cookie. Default is to use a session cookie.

Returns:**Example:**

```
//set a cookie with an expiration 6 months from now
var date = new Date();
date.setMonth(date.getMonth() + 6);
var success = s.Util.cookieWrite("my_cookie", "my_value", date);
```

Util.getQueryParam

Returns the value of a specified query string parameter, if found in the current page URL or in the provided string.

Because important data (such as campaign tracking codes, internal search keywords, etc.) is available in the query string on a page, `getQueryParam` helps capture the data into Analytics variables.

This utility replaces the [getQueryParam](#) plug-in.

Syntax:

```
s.Util.getQueryParam(key, [url], [delim])
```



Note: The syntax for the utility differs from the syntax for the plug-in.

Parameters:

Parameter	Description
key	(required) The name of the query string parameter that you want to get. This parameter is case sensitive.
url	(optional) Default url is <code>s.pageURL</code> or <code>window.location</code> . Specifying a value for this parameter overrides the URL from which the query parameter is retrieved to the one specified.
delim	(optional) Parameter delimiter in the URL. Default delimiter is "&". This lets you to specify an alternate query-string delimiter, such as ";".

Returns:

The function will return an empty string "" if no key is supplied, or if none of the URL options exists, or if the key is not found in the URL. If a fragment delimiter # is found in the URL, everything following the fragment delimiter is not considered. If the key exists and is assigned to a value, the value is URL decoded and returned.

Example:

```
s.doPlugins = function(s) {
  s.campaign = s.Util.getQueryParam("cid");
};
```

Offline Tracking

The following variables and functions let you store measurement calls when the application is offline.



Note: To enable offline tracking, your report suite must be timestamp-enabled. If timestamps are enabled on your report suite, your `trackOffline` configuration property must be `true`. If your report suite is not timestamp enabled, your `trackOffline` configuration property must be `false`. If this is not configured correctly, data will be lost. If you are not sure if a report suite is timestamp enabled, [contact Customer Care](#).

When enabled, Offline AppMeasurement behaves in the following way:

- The application sends a server call, but the data transmission fails.
- AppMeasurement generates a timestamp for the current hit.
- AppMeasurement buffers the hit data, and backs up buffered hit data to persistent storage to prevent data loss.

At each subsequent hit, or at the interval defined by `offlineThrottleDelay`, AppMeasurement attempts to send the buffered hit data, maintaining the original hit order. If the data transmission fails, it continues to buffer the hit data (This continues while the device is offline).

Property or Method	Description
trackOffline	Default: false Enables or disables offline tracking for the measurement library. Examples: <pre>s.trackOffline=true;</pre>
offlineLimit	Default: no limit Maximum number of offline hits stored in the queue.

Property or Method	Description
	<p>Examples:</p> <pre>s.offlineHitLimit=100;</pre>
offlineThrottleDelay	<p>Default: 0</p> <p>Specifies a cadence (or delay), in milliseconds, for sending buffered hit data when AppMeasurement detects an active network connection. Doing so mitigates the performance impact of sending multiple hits on the application.</p> <p>For example, if offlineThrottleDelay=1000, and it takes 300ms to send the hit data, AppMeasurement waits 700ms before sending the next buffered hit.</p> <pre>s.offlineThrottleDelay=1000;</pre>
forceOnline forceOffline	<p>Manually set the online or offline state of the measurement object. The library automatically detects when the device is offline or online, so these methods are needed only if you want to force measurement offline. <code>forceOnline</code> is used only to return to the online state after manually going offline.</p> <p>When measurement is offline:</p> <ul style="list-style-type: none"> • If <code>trackOffline</code> is true: hits are stored until measurement is online. • If <code>trackOffline</code> is false: hits are discarded. <p>Examples:</p> <pre>s.forceOffline(); s.forceOnline();</pre>

Data Collection

Data Collection Query Parameters

The following tables lists the query parameter that contains the value for each analytics variable that is sent to data collection.

This information can be used when debugging using [Packet Analyzers](#), when manually constructing image requests, or when using [Dynamic Variables](#).

Parameter	Analytics Variable	Report Populated	Description
aamlh	None	None	Audience Manager Location Hint (used in Experience Cloud Shared Profile integration)
aamb	None	None	Audience Manager Blob (used in Experience Cloud Shared Profile integration)
aid	None	None	Analytics visitor ID

Parameter	Analytics Variable	Report Populated	Description
AQB	None	None	Indicates the beginning of an image request.
AQE	None	None	Indicates the end of an image request, meaning the request was not truncated.
bh	None	Visitor Profile Technology Browser Height	Browser window height (in pixels)
bw	None	Visitor Profile Technology Browser Width	Browser window width (in pixels)
c	None	Visitor Profile Technology Monitor Color Depths	Color quality (in bits)
<code>c.[key]</code>	<code>s.contextData</code>	None	<p>Key-values pairs are specified in one of the following formats:</p> <pre><my.a>red</my.a></pre> <p>or:</p> <pre><my><a>red</my></pre> <p>Each of these examples result in a context data value of <code>my.a = red</code>. Multiple key-value pairs can be specified.</p> <p>In the query string, this context data variable would appear as <code>c.my.a=red</code></p>
c1-c75	<code>s.prop1-s.prop75</code>	All Custom Traffic reports	Traffic variables used in custom traffic reporting
cc	<code>s.currencyCode</code>	None	The type of currency used on the site
cdp	<code>s.cookieDomainPeriods</code>	None	Indicates the number of periods in a domain for cookie tracking; manually set.
ce	<code>s.charSet</code>	None	The character encoding of the image request
cl	<code>s.cookieLifetime (s_vi cookie lifetime in seconds)</code>	None	The lifetime of the visitor cookie.
ch	<code>s.channel</code>	Site Content Site Sections	The Site Sections variable used in traffic reporting
cp	Hit Type	Hit Type	Indicates whether the behavior is a result of direct

Parameter	Analytics Variable	Report Populated	Description
			interaction foreground or information the device is sending without direct interaction background.
ct	None	Visitor Profile Technology Connection Types	Connection Type (Modem, LAN, etc; can only populate in IE browsers)
D	dynamicVariablePrefix	None	See Dynamic Variables .
events or ev	s.events	Site Traffic Purchases, Shopping Cart, Custom Events	The commerce and custom events that occurred on the page; used in conversion reports
g	None	None	The current URL of the page, up to 255 bytes.
-g	None	None	URLs longer than 255 bytes are split, with the first 255 bytes appearing in the g parameter, with the remaining bytes appearing later in the query string in the -g= query parameter.
h1-h5	s.hier1-s.hier5	Site Content Hierarchy reports	Hierarchy variables; used in traffic reporting
hp	None	Visitor Profile Visitor Home Page	Indicates if current page is browser's home page (Y or N; can only populate in IE browsers)
j	None	Visitor Profile Technology Javascript Version	Shows the current Javascript version installed (generally 1.x)
k	None	Visitor Profile Technology Cookies	Are cookies supported in the browser (Y, N or U)
l1-l3	s.list1-s.list3	Custom Conversion	A delimited list of values that are passed into a variable, then reported as individual line items for reporting.
mid	None	None	Experience Cloud Visitor ID
ndh	None	None	Indicates whether the image request originated from JS file (1 or 0)
ns	s.visitorNameSpace	None	Specifies what domain the cookies are set on

Parameter	Analytics Variable	Report Populated	Description
oid	s.objectID	Site Content Links ClickMap	Object identifier for last page; used in ClickMap
ot	None	Site Content Links ClickMap	Object tag name for last page; used in ClickMap
p	None	Visitor Profile Technology Netscape Plug-Ins	Semicolon delimited browser plug-in names
pageName (or gn)	s.pageName	Site Content Pages	The page's designated name in reporting
pageType (or gt)	s.pageType	Site Content Pages Not Foun	Indicates whether it is a 404 page or not (Either 'error' or blank)
pccr	None	None	Only occurs for new visitors; prevents infinite redirects (Always true)
pe	s.linkType	Site Content Links Exit Links, File Downloads, Custom Links	Determines the type of custom link hit fired
pev1	None	Site Content Links Exit Links, File Downloads, Custom Links	URL the custom link hit occurred on
pev2	None	Site Content Links Exit Links, File Downloads, Custom Links	Custom link friendly name
pev3	None	All video reports	Used to track milestones in legacy video reporting; deprecated with v15
pf	None	None	For Adobe use only. Do not alter.
pid	None	Site Content Links ClickMap	Page identifier for last page; used in ClickMap
pidt	None	Site Content Links ClickMap	Page identifier type for last page; used in ClickMap
products (or pl)	s.products	Products Products	Products variable used in conversion reporting
purchaseID (or pi)	s.purchaseID	None	Used to deduplicate purchases, preventing revenue inflation
r	s.referrer	All traffic sources reports	Referring URL
s	None	Visitor Profile Technology Monitor Resolutions	Screen resolution (width x height)
server (or sv)	s.server	Site Content Servers	The page's server; used in traffic reporting

Parameter	Analytics Variable	Report Populated	Description
state	s.state	Visitor Profile Visitor State	Specifies the state as defined by the variable.
t	(automatic, sent with every hit that does not have a custom timestamp)	None	<p>The <code>t</code> parameter is in the following format:</p> <pre>dd/mm/yyyy hh:mm:ss D OFFSET</pre> <p>Where D is a number in the range 0-6 specifying the day of the week, and OFFSET represents:</p> <pre>offset from GMT in hours * 60 * - 1</pre> <p>For example:</p> <pre>23/09/2016 14:00:00 1 420</pre>
ts	timestamp	None	The custom timestamp calculated and sent in with the hit. Typically used for offline tracking.
v	None	Visitor Profile Technology Java	Java enabled (Y or N)
v0	s.campaign	Campaigns Tracking Codes	The campaign variable used in conversion reporting
v1-v75	s.eVar1-s.eVar75	All Custom Conversion reports	Conversion variables used in custom conversion reporting
vid	s.visitorID	None	The visitor's unique ID as set in the <code>visitorID</code> variable.
vmk	s.vmk	None	Visitor migration key; used to migrate from third-party to first-party cookies. Deprecated.
vvp	s.variableProvider	None	Used in Genesis integrations
xact	s.transactionID	None	The transaction ID used to link online data to offline data
zip	s.zip	Visitor Profile Visitor ZIP/Postal Code	Determines the zip code as defined by the variable


Parameter	Analytics Variable	Report Populated	Description
/5/ (for mobile protocol) or /1/ (for non-mobile protocol) in the image request URL.	None	None	Controls the order in which cookies and other methods are used to identify visitors.

Data Collection HTTP Headers

HTTP request and response headers are used to collect additional data beyond what is collected by AppMeasurement. This section describes the headers used during data collection.


HTTP Request Headers

Header	Usage
Cookie	Reading cookies previously created by our data collection servers. As of 2014, Adobe servers will discard all cookies that accompany a server call except those set by Adobe. See Cookies Used in the Experience Cloud for the full list of Adobe's cookies.
User-Agent	Used for browser, operating system, and mobile device detection.
X-Device-User-Agent	Used as an alternative to User-Agent for correct browser, operating system, and mobile device detection for some browsers like OperaMini.
X-Original-User-Agent	Used as an alternative to User-Agent for correct browser, operating system, and mobile device detection for some browsers like OperaMini.
X-OperaMini-Phone-UA	Used as an alternative to User-Agent for correct browser, operating system, and mobile device detection for some browsers like OperaMini.
X-Skyfire-Phone	Used as an alternative to User-Agent for correct browser, operating system, and mobile device detection for some browsers like OperaMini.
X-Bolt-Phone-UA	Used as an alternative to User-Agent for correct browser, operating system, and mobile device detection for some browsers like OperaMini.
UA-OS	Used as an alternative was to identify the operating system.
UA-Pixels	Used as an alternate source for the screen resolution of the client screen.
UA-Color	Used as an alternate source for the color depth of the client screen.
X-moz	Detecting that the data collection request was made as part of pre-fetching a webpage.
X-Purpose	Detecting that the data collection request was made as the browser was showing a preview of a webpage.
Accept	Used to identify the image formats supported by the browser so we know if we need to send back a GIF or WBMP image.
Referrer	Used as a fallback for getting information about the page URL the data collection request was made from when it wasn't passed in on the query-string or when it's different from the value in the query-string.
X-Forwarded-For	Used to find the correct IP address for the client that made the data collection request. The IP address is used to generate geographic region, mobile carrier, and other reports.

 **Note:** Implementations using dynamic variables have the option of reading in other HTTP request headers not listed above.

HTTP Response Headers

Header	Usage
Access-Control-Allow-Origin	Used to enable support for cross-origin resource sharing style data collection requests to our servers.
Expires	Browser caching control.
Last-Modified	Browser caching control.
Cache-Control	Browser caching control.
Pragma	Browser caching control.
ETag	Browser caching control.
Vary	Browser caching control.
P3P	Provides the default or custom P3P policy for the data collection request.
Status	Contains "SUCCESS" or "FAILURE" status for a no content request. Used only when the request specifies that no content should be returned.
Reason	Contains the reason for the failure status of a no content request. Used only when the request specifies that no content should be returned.
Location	Used to redirect the client making the data collection request off to a different URL. An example is our cookie handshake to detect the ability to set the visitor ID cookie.
Content-Type	Specifies the type of content sent back to the client (GIF, text, Javascript, etc).
Content-Length	Specifies the size of the content sent back to the client.

 **Note:** Other HTTP headers may be set in the response for internal status monitoring. Some of these headers might be returned to the browser, but it is not necessary that they receive them.

Variable Overrides

Variable overrides let you change a variable value for a single track or track link call.

To override variables, create a new object, assign variable values, and pass this object as the first parameter to `s.t()`, or as the fourth parameter to `s.tl()`:

```
s.eVar1="one";
s.eVar2="two";
s.eVar3="three";

overrides = new Object();
overrides.eVar1="1_one";
overrides.eVar2="";

s.t(overrides);
// values passed: eVar1="1_one", eVar2="", eVar3="three"

s.linkTrackVars="eVar1,eVar2,eVar3,events";
s.eVar1="one";
s.eVar2="two";
s.eVar3="three";
```

```

overrides = new Object();
overrides.eVar1="1_one";
overrides.eVar2="";

s.tl(this,'e','AnotherSite',overrides,'navigate')
// values passed: eVar1="1_one", eVar2="", eVar3="three"

```

Report Suite IDs - Dynamic Accounts

The .js file can be configured to automatically select a report suite ID. The .js file automatically sends the image request to the report suite based on the URL. For example, if the URL is `www.mysite.com`, the image request is automatically sent to report suite A. If the URL is `www.mysite1.com`, the image request is automatically sent to report suite B.

These strings can be found within any of the following:

- Host/domain (default setting)
- Path
- Query String
- Host/domain and Path
- Path and Query String
- Full URL

For more information on configuring Analytics to automatically select a **Report Suite ID**, contact Adobe Live Support.

Defining the URL Segment to Match

Given the following sample URL, the portions of the URL are shown below, along with the **s.dynamicAccountMatch** variable that must be set. (The default - if **s.dynamicAccountMatch** is not defined - is to search the Host/Domain Name only).

Sample URL: `http://www.client.com/directory1/directory2/filename.html?param1=1234¶m2=4321`

Portion	Example (from above)
Host/Domain Name	<code>www.client.com</code>
Path	<code>directory1/directory2/filename.html</code>
Query String	<code>param1=1234&param2=4321</code>
Host/Domain and Path	<code>www.client.com/directory1/directory2/filename.html</code>
Path and Query String	<code>directory1/directory2/filename.html?param1=1234&param2=4321</code>
URL	<code>http://www.client.com/directory1/directory2/filename.html?param1=1234&param2=4321</code>
Portion	<code>s.dynamicAccountmatch</code>
Host/Domain Name	Undefined
Path	<code>window.location.pathname</code>
Query String	<code>(window.location.search?window.location.search:"?")</code>
Host/Domain and Path	<code>window.location.host+window.location.pathname</code>
Path and Query String	<code>window.location.pathname+(window.location.search?window.location.search:"?")</code>
URL	<code>window.location.href</code>

Consider the following example:

- `s.dynamicAccountSelection=true`
- `s.dynamicAccountList="devreportsuite1=qa.client.com;reportsuite2=clientdirectory;reportsuite1=client.com"`
- `s.dynamicAccountMatch=window.location.host+window.location.pathname`

Multiple Rules

If multiple rules are selected (see example above), the rules are executed from left to right. The rules stop as soon as a match is made, as shown below (with the given set of rules).

- `s.dynamicAccountSelection=true`
- `s.dynamicAccountList="devreportsuite1=qa.client.com;reportsuite1=client.com"`

The code first checks to determine if "qa.client.com" exists within the Host/Domain Name. If so, the report suite "devreportsuite1" is selected, and the match stops. Separate multiple rules with semi-colons.

Default Report Suite

The `s_account` variable can be set first, and acts as a default value in case any of the specified strings cannot be found. Below is an example:

```
var s_account="defaultreportsuiteid"
s.dynamicAccountSelection=true
s.dynamicAccountList="devreportsuite1=qa.client.com;reportsuite1=client.com"
```

In the case above, if the host/domain name did not contain either "qa.client.com" or "client.com," the report suite "defaultreportsuiteid" would be used.

Common Errors

Common errors in dynamic accounts are described in the following sections.

Hard Coded Account

If the desire is to always send data to a specific report suite, set **s_dynamicAccountSelection** to false (alternately, the variables may be removed altogether):

```
var s_account="defaultreportsuiteid"
REMOVE: s.dynamicAccountSelection=true
REMOVE: s.dynamicAccountList="devreportsuite1=qa.client.com;reportsuite1=client.com"
```

In the case above, defaultreportsuiteid is always used after the other two lines are removed.

Placement of Code

Defining `s_account` after the lines of code does not override the dynamic account selection, as shown below.

```
var s_account="defaultreportsuiteid"
s.dynamicAccountSelection=true
s.dynamicAccountList="devreportsuite1=qa.client.com;reportsuite1=client.com"
s_account="anotherreportsuiteid"
```

In the example above, the account "anotherreportsuiteid" overrides "defaultreportsuiteid," but does not override any matches that occur in **s.dynamicAccountList**. The function that evaluates **s.dynamicAccountList** is actually executed much later in the .JS file.

Multi-Suite Tagging

Multi-suite tagging may be used in conjunction with dynamic account selection, as shown below.

```
s.dynamicAccountSelection=true
s.dynamicAccountList="suiteid1,suiteid2=client.com"
```

Dynamic Account Match

Do not put the **dynamic account match** variables in quotes. The options are displayed below.

Host/Domain Name	None
Query String	s.dynamicAccountMatch=(window.location.search?window.location.search:"?")
Host/Domain and Path	s.dynamicAccountMatch=window.location.host+window.location.pathname
Path and Query String	s.dynamicAccountMatch=window.location.pathname+(window.location.search?window.location.search:"?")
Full URL	s.dynamicAccountMatch=window.location.href

Collecting Data From Form Elements

You can capture the values of form elements, such as radio button and checkbox items, into reports. This helps you analyze the most popular choices in your online forms.

For example, if you had a radio button letting the user specify his or her favorite genre of music (such as rock, rap, classical, or jazz), you could capture this response in a variable to determine the overall music preferences of your user base.

The best way to capture this data depends on how your forms are processed. It also depends on whether the form selections you want to capture are available in the query string on the page following the form submission. Examples in this article are given in PHP. However, you can adapt these concepts to another language, depending on your server environment.

This information is suited for advanced users who are well-versed in both reporting and implementation. Do not attempt to make any edits to your implementation without complete knowledge of its consequences. If you require implementation changes, contact your organization's Account Manager.

GET Method

If your form uses a **GET** method to submit data, you have access to the desired data in the query string of the URL on the page following form submission. You can use the **getQueryParam** plug-in to capture this data out of the query string automatically and place it into the variable of your choosing.

POST Method

If your form uses a **POST** method to submit data (which is more common), you have the results for each specific form element available to you in the **\$_POST superglobal**. To capture this in a variable, you want to determine the form element name in question. Using the music genre example mentioned before, part of the form element in question look like this:

```
<input type="radio" name="music_genre" value="rock">
```

This radio button belongs to the "music_genre" form element. You then have access to the user's selected value by using `$_POST['music_genre']`. This can be written to a variable on the page following the form submission:

```
s.eVar1="<?=$_POST['music_genre'];?>"
```

The **eVar1** variable receives a copy of whatever value was submitted to your server through the form, as specified in the `value=` property.

If you need additional information regarding this custom implementation method, contact your organization's Account Manager. They can arrange a meeting with one of our Implementation Consultants to provide the help you need.

Tracking Across Different Implementation Types

This information is intended for advanced users who are well-versed in both reporting and implementation. Do not attempt to edit your implementation without understanding the consequences. If you require implementation changes, contact your organization's Account Manager.

If you use more than one type of implementation (such as JavaScript and hardcoded image requests), ensure that the following variables are specified and match each other:

- `s_account`
- `s.visitorNamespace`
- `s.trackingServer`
- `s.trackingServerSecure` (if using SSL)

If each of these do not match across implementations, users may be tracked as separate visitors.

Implementation Guidelines

Following these guidelines results in using the same cookie domains, which lets visits be tracked between various types of implementations.

- **RSID:** The report suite ID
- **VNS:** Visitor name space, the subdomain of `2o7.net` or `omtrdc.net` used to store the **visitor ID** cookie
- **COOKIEDOMAIN:** Your VNS + trackingServer. Depending on your data center and RDC configuration, these can greatly vary. See [Regional Data Collection](#) or [contact Client Care](#) if you are unsure about you data collection domain.

Javascript:

```
var s_account="RSID"
s.visitorNamespace="VNS"
s.trackingServer="VNS.COOKIEDOMAIN.net"
Hardcoded image request:
```

AppMeasurement:

```
var s_account="RSID"
s.visitorNamespace="VNS"
s.trackingServer="VNS.COOKIEDOMAIN.net"
Hardcoded image request:
```

Hardcoded image request:

```

<!-- Note that the visitor namespace is defined twice in hardcoded image requests; once in the http subdomain, and another using the ns= query string parameter! -->
```

If using a first-party cookie implementation, `VNS.COOKIEDOMAIN.net` can be replaced with the first-party cookie domain used. For example, first-party cookies on `adobe.com` would be replaced with something similar to `metrics.adobe.com`.

Implementation Example

Using `adobe.com` as an example, the implementations described here reference the same **visit** cookie.

Javascript:

```
var s_account="omniturecom"
s.visitorNamespace="omniture"
s.trackingServer="omniture.112.2o7.net"
```

Hardcoded image request:

```

```

Appmeasurement:

```
s.account="omniturecom";  
s.visitorNamespace="omniture";  
s.trackingServer="omniture.112.2o7.net";
```

And if first-party cookies are utilized:

Javascript:

```
var s_account="omniturecom"  
s.visitorNamespace"omniture"  
s.trackingServer="metrics.omniture.com"
```

Hardcoded image request:

```

```

Appmeasurement:

```
s.account="omniturecom";  
s.visitorNamespace="omniture";  
s.trackingServer="metrics.omniture.com";
```

AJAX-Tracking Rich Media Applications

AJAX is an emerging concept in web design that uses multiple technologies to create and manage dynamic content on Web pages.

The need to track user interaction with **AJAX** and other rich media applications is paramount to analytics success and realization of the return on investment in the Web design. The focus of this white paper is to provide recommendations for tracking rich Internet applications, specifically those that use **AJAX**.

Rich Internet Applications (RIA)

Rich Internet Applications (RIAs) are changing the face of the Web. They bridge the gap between the promise of on-demand technologies for the masses and realistic user experiences. RIAs have been maturing for years. The biggest leaps forward have occurred with wide-scale adoption of browsers, which support the underlying technologies that power these applications. While RIA has many forms and supporting technologies, the most common, and perhaps most widely adopted, are AJAX and Flash. It is important to understand that the technology is not what defines an RIA, but its usability and application.

What to Track

One of the most commonly asked questions with RIA is how to track micro-level activity separate from macro-level activity, and when it is appropriate to do either. For example, say you have an application that allows customers to uniquely configure a product. The application may have significant steps the users are exposed to. Are these steps considered page views? In addition, there are micro-level activities within each step. Should these activities be tracked as page views?

What if you want to understand the flow between activities, or which features get the most activity? The trouble with RIA is that each application is unique. They are designed to mimic realistic actions, and since actions are relative to the situation, the possibilities are endless. However, most applications have a few major components: milestone

steps, features, and micro-level actions within features. So, while each application requires some consideration as to what specifically should be measured, there are some generalizations that can be applied to RIA tracking.

Macro-Level Activity

Macro-level activity usually constitutes the loading of the application. This provides information on visits, visitors, instances, value to future actions, and so forth. It can, and should, also represent major steps in the process. A good rule of thumb is that if an RIA action changes the application more than 50% (or whatever is considered significantly changing the user experience or content), then it is macro-level, and should be tracked as a page view.

Micro-Level Activity

Micro-level activity includes any changes less than 50% (or not considered as significantly changing the user experience or content). Toggling between color selections, for instance, would be considered micro-level activity. Adobe recommends that micro-level tracking be related to features. For example, in the case of toggling between colors, is it really important to understand which colors were considered? Or is it more important to know that the color selection feature was used? Perhaps both are important, and if so, capture both, but when measuring the effectiveness of RIA, consider the feature level activity as being more valuable.

All micro-level activity should be tracked as custom links with specifics measured through associated traffic variables (props and eVars if the use needs to be measured against success events). This ensures that page views are not inflated by micro-level activity, and allows for path analysis through the traffic variable.

What to Analyze

It is important to understand how effectively your RIA is driving success. Success is most commonly measured through conversions. A macro-level analysis provides insight into RIA effectiveness as a whole. Micro-level analysis may provide insight into which features help drive conversion.

You should measure efficiency of your RIA. This is an analysis of micro-level activity relative to the RIA macro metrics. Do users go through more steps than necessary to arrive at the same goal? Analysis metrics might include visits/features activity; page views/feature activity, visitors/feature activity, and so forth.

Conduct analysis on path flow and fall out. Are users avoiding the RIA and finding another path to the goal? Run fallout reports built around the site and RIA flow. Run path analysis from landing pages to gauge the true traffic patterns. Look at barriers and incentives to guide users toward the goal.

Suggested Metrics

- RIA Visits
- RIA Visitors
- RIA Page Views
- RIA Feature Activity (Custom Links) measure click activity by feature

Suggested Analysis

- RIA Feature Activity / RIA Page Views
- RIA Feature Activity / RIA Visits -
- RIA Page Views / Success Metric - Conversion Ratio: measures application effectiveness
- Total RIA Activity / Success Metric - Conversion Ratio: measures application efficiency
- Feature RIA Activity / Success Metric - Conversion Ratio: measures application feature efficiency
- Path Flow to and from RIA
- Fallout Rates through RIA conversion process

Implement with AJAX

Implementing with **AJAX** is exactly like deploying code on a standard HTML page.

The business has questions that need answers, the needs are assessed and variables assigned. The design is then applied and deployed. These concepts should be familiar if you have already been through the initial stages of implementation.

Design the Solution

The difference when throwing **AJAX** into the mix is first understanding the level of detail that needs to be gathered. The potential of content changing on the page (macro-level) or tracking attributes of the application (micro-level) determines which variables need to be set and which method of sending data to Adobe works best.

Deploy the Code

There are two functions in the JavaScript code that allow you to send data. There are some distinct guidelines that should be followed to know which method should be used to send data.

Collect Macro-Data (Page)

The `t()` function in the Adobe code sends a standard format image request, incrementing total site page views. All Adobe variables that have been assigned values send data. The primary focus of using this function within RIAs revolves around the value of the `pageName` variable. You should use this function when:

- The new content is considered a page view for the site or is considered moving from one page to another.
- The content of the page changes more than 50% (or whatever is considered significantly changing the user experience or content).
- The page path must be tracked for each user interaction with the RIA.

Send Macro-Data (Page)

If an image request was previously made on the same page, you must first clear the values of the previously-set variables. Use the `clearVars()` function in AppMeasurement for JavaScript, or write a simple JavaScript function to clear the variables if you are using H code. Set the values appropriate for the changed content, namely the `pageName` variable. After the variables are set call the `t()` function.

Example



Note: Before you call `s.t()`, you must clear any values on the `s` object that you do not want to persist. If you are using AppMeasurement for JavaScript, you can call `s.clearVars()`. If you are using H code, write a simple routine to set variables to an empty string.

```
s.clearVars();
s.pageName="New Page"
s.prop1="some value"
void(s.t());
```

The following example shows a tracking call in the `done` callback of the JQuery `.ajax` function:

```
$.ajax({
  url: "test.html",
  dataType: "html"
})
.done(function( response ) {
  $( "#content" ).html( response );
  s.clearVars();
  s.pageName = $( "h1:first" ).text();
```

```
s.tl();
});
```

Collect Micro-Data (Link)

The `tl()` function does not send the `pageName` variable and is selective in the other variables that are sent. Because of this restrictive design, it is well-suited for tracking micro-level statistics such as button clicks, link clicks, scrolling, and other similar events. You should use this function when:

- You do not include a new page name or inflate page view statistics for the site.
- The tracking of user interactions with buttons, links, or specific features within a page/application is desired.
- Track the interaction path of the application.

Path analysis can be enabled for traffic variables (not restricted to page pathing) which is a useful tool in analyzing the order users interact with your page or application. For example, you see that a user interacted first with `button1` and then clicked on `button4` before leaving the application. This link-level path can be traced without inflating page-level statistics by using the `tl()` function.

Send Micro-Data (Link)

If an image request was previously made on the same page, clear the values of the previously-set variables. This can be accomplished by:

- Writing a simple JavaScript function to clear the Adobe variables.
- Set the `linkTrackVars` and `linkTrackEvents` variables if you have not already done it in the `s_code.js` file.
- Set the values appropriate for the changed content, namely the `pageName` variable.
- After the variables are set, call the `tl()` function.

Syntax

```
//set linkTrackVars and linkTrackEvents> (if applicable)
//set new variables
s.tl(this, 'o', 'Link Name');
```

Example

```
s.linkTrackVars="prop1,eVar1,events"; s.linkTrackEvents="event1";
s.prop1="some value"; s.eVar1="another value"; s.events="event1";
s.tl(this, 'o', 'My Link Name');
```

Placement of Code

There are generally two places to track data with **AJAX**: at the time of the request or in the reply. In most cases, macro-data (page information) should be sent at the time of reply by having the code embedded in the HTML of the new content. For micro-data tracking (links, etc.) it is more common to use a custom links approach by inserting the code in the `onClick` attribute of the link, button, etc.

External Email Tracking

Companies use Analytics to determine the success of an email campaign.

Analytics can report email campaign analysis data in several key metrics, including the following:

Metric	Description
Click-throughs	Displays the number of click-throughs tracked from the email to the landing page.
Purchases and/or Successes	Displays the number of purchases resulting from the email.

Metric	Description
Orders	Displays the number of orders placed as a result of the email.
Yield	Displays the dollar amount per visit generated from the email.
Conversion	Displays the number of leads, registrations, or any other success event generated from the email.

Modifications to the HTML email body and the JavaScript library are required in order to capture the key metrics shown above.

Implementation

There are several steps to follow in order to successfully display email campaign analysis data. The steps are described as follows:

1. Create unique tracking codes.

Often, users ask for tracking recommendations for each unique campaign. This is entirely up to them, based on what works best. Each user is different. Adobe recommends that each user generate friendly tracking codes, as shown in the example below:

- `sc_cid=A1123A321` > "A" flags affiliate campaign
- `sc_cid=EM033007` > "EM" flags email campaign
- `sc_cid=GG987123` > "GG" signifies Google and is a paid search campaign

Contact Adobe Customer Care for details on setting up and using tracking codes.

2. Add query string parameters to HTML email links.

In order to track a user click-through and subsequent success events, a query string parameter needs to be added to each link within the HTML email. You can choose to track each link separately or track all links together. Each link can have a unique tracking code, or all links can have the same tracking code. Consider the following hypothetical link within the email to a website:

```
<a href="http://www.mycompany.com/index.asp">Visit our home page</a>
```

The following query string parameters `?sc_cid=112233B` should be added to the link above:

```
<a href="http://www.mycompany.com/index.asp?sc_cid=112233B">Visit our home page</a>
```

3. Update the JavaScript library.

Altering code in the JavaScript file, `s_code.js`, lets you capture how many users (and which users) clicked-through from the email and participated in subsequent success events. There are two steps to updating the JavaScript library.

a. Customize `s_code.js` by calling `getQueryParam`.

The `s_code.js` file should be placed in a location on the Web server where each Web page can access it. The `doPlugins` function within this file should be altered so it captures the query string parameters on the email links. For example:

```
/* Plugin Config */
s.usePlugins=true
function s_doPlugins(s) {
  /* Add calls to plugins here */
  // External Campaigns
  s.campaign=s.getQueryParam('source')
}
s.doPlugins=s_doPlugins
```

Each query string parameter that needs to be copied into a variable should have one **getQueryParam** call. In the example above, the query string parameter **sc_cid** is copied into *campaign*.

Only the first call to **getQueryParam** is required to capture click-throughs. Contact Adobe Customer Care to implement this function and ensure that your version of the JavaScript file contains the **getQueryParam** plug-in.

- b. Make sure the Code to Paste JavaScript tags are on all landing pages. This Code to Paste must reference the version of `s_code.js` altered in Part A.

The following points are important to remember when updating the JavaScript library. These points are listed below.

- The query string parameter **sc_cid** must be visible in the URL on the final landing page otherwise no click-through conversion is recorded.
- The **sc_cid** parameter is an example of a query string parameter. Any query string parameter can be used and captured by the **getQueryParam** plug-in. Make sure that the query string parameters are used only for campaign tracking. Any time the parameters appear in a query string, their values are copied into *campaign*.

4. Use **SAINT** to classify campaign tracking codes.

The **SAINT Campaign Management Tool** can be used to convert tracking codes into user-friendly names. It can also be used to summarize the success of each email campaign. Step 5, below, outlines the process required to set up an email campaign.

5. See pathing by email campaign (optional).

Pathing analysis by email campaign can be accomplished similarly to pathing by another campaign. You can use a variable to show pathing by campaign, as explained in the following steps:

- a. Consult Adobe Customer Care about turning on pathing for a **Custom Insight** variable (prop)
- b. On all pages, copy the page name into the designated `s.prop`.
- c. On the email landing page, append the name of the email campaign to the prop. The result displays as shown below:

```
s.prop1="Home Page : 123456"
```

When pathing is enabled for the **Custom Insight** variable, you can use **Path** reports (such as **Next Page Flow** or **Fallout**) to see visitor navigation from the landing page.

Implementing Adobe Opt-Outs

Some visitors to your website may prefer not to have their browsing information aggregated and analyzed by Adobe Experience Cloud products and services or used to provide relevant content and advertising. Adobe offers you the ability to provide visitors to your website a means to opt out of their information being collected by the following Adobe products:

- Adobe Analytics
- Adobe Target
- Adobe Audience Targeted Creative
- Adobe AudienceManager
- Adobe AdLens
- Adobe Experience Manager

Privacy policy recommendations

Adobe recommends that you provide your website visitors with easy-to-find and easy-to-understand information regarding the ability to opt out of having their browsing information collected by Adobe products or services.

Visitors can learn more about how Adobe generally uses information it collects in connection with providing our products and services to our customers in the [Adobe Privacy Center](#). However, because you exclusively control how to implement our services on your web sites, it is up to you to describe to your website visitors the specific ways in which they use our products and services. You are responsible for the creation of your own privacy policy, for complying with your privacy policy, for complying with your service agreement with Adobe, and for complying with all applicable laws.

Opt-outs for Adobe Analytics (including reports & analytics, data warehouse, ad hoc analysis and SearchCenter+)

Adobe offers three types of opt-outs for Adobe Analytics (including reports & analytics, data warehouse, ad hoc analysis and SearchCenter+):

- If you implement Adobe Analytics products with your own first-party cookie, you need to [develop your own customized opt-out link](#) for your website visitors.
- Adobe also provides an opt-out mechanism to the public for websites using cookies set from Adobe's 2o7.net and omtrdc.net domains. This opt-out mechanism can be accessed from the [Adobe Privacy Center](#).
- Your customers have the option of enabling opt-out using the browser's cookie settings. See [Enable privacy settings for browser cookies](#).

Regardless of the opt-out mechanism you choose, Adobe recommends that you clearly describe the availability of the opt-out mechanism in your privacy policy or as otherwise required by law or recommended according to current best practices.

Add an Opt-Out Link

Specify an opt-out link, and customize the branding for the link. Visitors to your web site may choose not to have their activity tracked in Adobe's analytics products by visiting the opt-out page for your data collection domain.

If a user chooses not to be tracked and an opt-out cookie is set, your JavaScript file will continue to send data to Adobe servers but that data will not be processed or reported on.

The `collection_domain` section of the URL structure is the trackingServer used in your JavaScript file. The collection domain used for your Adobe Analytics implementation can be seen in the DigitalPulse debugger in the first row of the Adobe Analytics table, which is labeled either "First Party Cookies" or "Third Party Cookies" depending on your implementation. The collection domain for your website may contain 2o7.net, omtrdc.net or your website domain, such as `metrics.example.com`.

Visitors opt-out by clicking the link on the opt-out page, thereby causing a cookie to be set in their browser. By having the `omnitur_optout` cookie for the applicable tracking domain, the user's activities will not be reported by Adobe Analytics. You can provide your own link to the opt-out cookie, or you can follow the steps below to set the opt-out cookie.

Adobe offers opt-outs for all implementation types. You are responsible for your own privacy policy and for remaining in compliance with your signed terms. Note that the link to the opt-out page changes based on your implementation type, as outlined here.

If you implement Adobe Analytics products and services with cookies set on domains owned by Adobe (i.e. 207.net or omtrdc.net), you can point your website visitors to the opt-out mechanism provided in the [Adobe Privacy Center](#)

for all sites that use Adobe cookies for Adobe Analytics products and services. The direct link to the Adobe opt-out mechanism is http://collection_domain/optout.html.

More information about Adobe Analytics privacy practices can be found at <http://www.adobe.com/privacy/advertising-services.html>.

- [Opt-out Page URL Structure](#)
- [Example Opt-Out URLs](#)
- [Branding your Opt-Out URL](#)

Opt-out Page URL Structure

Your opt out page is at the following URL:

```
http://collection_domain/optout.html[?optional_parameters]
```

The `optional_parameters` include:

`locale=[code]`: Provides a translated version of the opt-out page. The following locales are supported:

- en_US (default)
- de_DE
- es_ES
- fr_FR
- jp_JP
- ko_KR
- zh_CN
- zh_TW

`popup=1`: Treats the page as if it were a popup, and offers a “Close Window” button.

Example Opt-Out URLs

An English web page in a full window containing a link that, when clicked, will prevent the visitor from being tracked on `metrics.example.com`:

```
http://metrics.example.com/optout.html
```

A French web page in a full window, containing a link that, when clicked, will prevent the visitor from being tracked on `example.d3.sc.omtrdc.net`:

```
http://example.d3.sc.omtrdc.net/optout.html?locale=fr_FR
```

A German web page, in a popup window, containing a link that, when clicked, will prevent the visitor from being tracked on `example.112.2o7.net`:

```
http://example.112.2o7.net/optout.html?popup=1&locale=de_DE
```

Branding your Opt-Out URL

You can provide a link, such as the following somewhere on your website:

```
<a href=" http://stats.adobe.com/optout.html?optout=1&confirm_change=1 ">  
Click Here to Opt Out! </a>
```

Where `stats.adobe.com` is replaced with whatever the `s.trackingServer` variable is set to.

Additionally, if you want like to provide a link to opt-in, use the same URL, but replace `?optout=1` with `?optin=1`, and keep the `confirm_change=1`.

Implementation Plug-ins

AppMeasurement for JavaScript plug-ins are programs or functions that perform several advanced functions.

These plug-ins extend the capabilities of your JavaScript file to give you more functionality that is not available with a basic implementation. Adobe offers a number of other plug-ins as part of advanced solutions. Contact your Account Manager if you want to capture data using JavaScript but are unsure how to proceed.

Calling Plug-ins with *doPlugins* Function

JavaScript plug-ins are usually called by the *doPlugins* function, which is executed when the *t()* function is called in the **Code to Paste**.

Consequently, if you set a variable in the *doPlugins* function, you can overwrite a variable you set on the HTML page. The only time the *doPlugins* function is not called is when the **usePlugins** variable is set to 'false.'

Code Example

The code example below is what the *doPlugins* function looks like in your JavaScript file:

AppMeasurement for JavaScript:

```
/* Plugin Config */
s.usePlugins=true
s.doPlugins=function(s) {
  /* Add calls to plugins here */
}
```

H code:

```
/* Plugin Config */
s.usePlugins=true
function s_doPlugins(s) {
  /* Add calls to plugins here */
}
s.doPlugins=s_doPlugins
```



Note: H code and earlier versions use a different syntax to support some very old browsers (such as IE 4 and 5).

Renaming the *doPlugins* Function

The *doPlugins* function is typically called *s_doPlugins*. In certain circumstances, (usually when more than one version of code may appear on a single page) the *doPlugins* function name may be changed. If the standard *doPlugins* function needs to be renamed to avoid conflicts, ensure that *doPlugins* is assigned the correct function name, as shown in the example below.

```
/* Plugin Config */
s_mc.usePlugins=true
function s_mc_doPlugins(s_mc) {
  /* Add calls to plugins here */
}
s_mc.doPlugins=s_mc_doPlugins
```

Using *doPlugins*

The *doPlugins* function provides an easy way to give default values to variables or to take values from **query string parameters** on any page of the site. Using *doPlugins* is often easier than populating the values in the HTML page

because only one file must be updated. Keep in mind that changes to the JavaScript file are not always immediate. Return visitors to your site are often using cached versions of the JavaScript file. This means that updates to the file may not be applied to all visitors for up to one month after the change is made.

The following example shows how the *doPlugins* function can be used to set a default value for a variable and to get a value from the query string.

```
/* Plugin Config */
s.usePlugins=true
s.doPlugins=function(s) {
  /* Add calls to plugins here */
  // if prop1 doesn't have a value, set it to "Default Value"
  if(!s.prop1)
  s.prop1="Default Value"

  // if campaign doesn't have a value, get cid from the query string
  if(!s.campaign)
  s.campaign=s.getQueryParam('cid');

  // Note: The code to read query parameters is different for
  // Appmeasurement for JavaScript since a plug-in is not required:
  // s.campaign=s.Util.getQueryParam('cid');
}
```

Installed Plug-ins

To find out whether a plug-in is included in your JavaScript file and ready for use, look in the **Plugins Section** of the JavaScript file. The following example shows the **getQueryParam** function.

```
/****** PLUGINS SECTION *****/
/* You may insert any plugins you wish to use here. */
/*
 * Plugin: getQueryParam 1.3 - Return query string parameter values
 */
s.getQueryParam=new Function("qp","d",""
+"var s=this,v='',i,t;d=d?d:'';while(qp){i=qp.indexOf(',');i=i<0?qp.l"
//
// ... more code below ...
//
```

s.abort flag

The abort flag can be set inside doPlugins to cause the current track call to not be sent.

The abort flag is reset with every tracking call, so if a subsequent tracking call also needs to be aborted the flag will need to be set again inside doPlugins.

```
s.doPlugins = function(s) {
  s.campaign = s.getQueryParam("cid");
  if ((!s.campaign) && (!s.events)) {
    s.abort = true;
  }
};
```

This lets you centralize the logic you use to identify activity that you do not want to track, such as some custom links or external links in display ads.

appendList

The apl (or appendList) plug-in lets you append a value to any delimited lists, with the option of a case-sensitive or case-insensitive check to ensure that the value does not already exist in the list. The APL plug-in is referenced by several standard plug-ins but can be used directly in a variety of situations.

This plug-in is useful for:

- Adding an event to the current events variable
- Adding a value to a list variable without duplicating a value in the list
- Adding a product to the current products variable based on some page logic
- Adding values to the parameters *linkTrackVars* and *linkTrackEvents*

Use Case 1

Scenario	Add <i>event1</i> to the current events variable while ensuring the event isn't duplicated. <code>s.events="scCheckout"</code>
Code	<code>s.events=s.apl(s.events,"event1",",",1)</code>
Results	<code>s.events="scCheckout,event1"</code>

Use Case 2

Scenario	Add the value <i>history</i> to the list variable <i>prop1</i> , with <i>history</i> and <i>History</i> considered the same value. <code>s.prop1="Science,History"</code>
Code	<code>s.prop1=s.apl(s.prop1,"history",",",2)</code>
Results	<code>s.prop1="Science,History"</code> <i>history</i> is not added because <i>History</i> is already in the list.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Implementation

Follow these steps to implement the APL plug-in.

1. Request the plug-in code from Customer Care or your currently assigned Adobe consultant.
2. Add call(s) to the API function as needed within the `s_doPlugins` function

Here is how the code might look on your site:

```
/* Plugin Config */
s.usePlugins=true
function s_doPlugins(s) {
/* Add calls to plugins here */
s.events=s.apl(s.events,"event1",",",1)
}
```

```
s.doPlugins=s_doPlugins
```

Supported Browsers

This plug-in requires that the browser supports JavaScript version 1.0.

Plug-in Information

Plug-in Information	Description
Parameters	<p>apl((L,v,d,u)</p> <p>L= source list, empty list is accepted</p> <p>v = value to append</p> <p>d = list delimiter</p> <p>u (optional, defaults to 0) Unique value check. 0=no unique check, value is always appended. 1=case-insensitive check, append only if value isn't in list. 2=case-sensitive check, append only if value isn't in list.</p>
Return Value	original list, with appended value if added
Usage Examples	s.events=s.apl(s.events,"event1","",1);

The source list L can be an empty list, such as L="". The returned value will either be an empty list, or a list of one value.

Plug-in Code

```

/*****
 *
 * Main Plug-in code (should be in Plug-ins section)
 *
 *****/
/*
 * Plugin Utility: apl v1.1
 */
s.apl=new Function("l","v","d","u","
+"var s=this,m=0;if(!l)l='';if(u){var i,n,a=s.split(l,d);for(i=0;i<a."
+"length;i++){n=a[i];m=m|(u=1?(n==v):(n.toLowerCase()==v.toLowerCase"
+"e()));}}if(!m)l=l?l+d+v:v;return l");

/*****
 *
 * Commented example of how to use this is doPlugins function
 *
 *****/

Not Applicable - Utility function only

/*****
 *
 * Config variables (should be above doPlugins section)
 *
 *****/

None

```

```

/*****
 *
 * Utility functions that may be shared between plug-ins (name only)
 *
 *****/

s.split

```

doPlugins Function

JavaScript plug-ins are usually called by the `doPlugins` function, which is executed when the `t()` function is called in the Code to Paste.

Consequently, if you set a variable in the `doPlugins` function, you may overwrite a variable you set on the HTML page. The only time the `doPlugins` function is not called is when the `usePlugins` variable is set to `false`.

Code Example

The `doPlugins` function is typically called `s_doPlugins`. However, in certain circumstances (usually when more than one version of Analytics code may appear on a single page), you can change the `doPlugins` function name. If the standard `doPlugins` function needs to be renamed to avoid conflicts, assign `doPlugins` the correct function name, as shown in the example below.

```

/* Plugin Config */
s_mc.usePlugins=true
function s_mc_doPlugins(s_mc) {
/* Add calls to plugins here */
}
s_mc.doPlugins=s_mc_doPlugins

```

Using doPlugins

This function provides an easy way to give default values to variables, or to take values from query string parameters on any page of the site. Using `doPlugins` can be easier than populating the values in the HTML page, because only one file must be updated. Changes to the JavaScript file are not always immediate. Return visitors to your site are often using cached versions of the JavaScript file. Meaning, updates to the file may not be applied to all visitors for up to one month after the change is made.

The following examples show how you can use the `doPlugins` function to set a default value for a variable and to get a value from the query string.

```

/* Plugin Config */
s.usePlugins=true
function s_doPlugins(s) {
/* Add calls to plugins here */
// if prop1 doesn't have a value, set it to "Default Value"
if(!s.prop1)
s.prop1="Default Value"

// if campaign doesn't have a value, get cid from the query string
if(!s.campaign)
s.campaign=getQueryParam('cid');
}
s.doPlugins=s_doPlugins

```

Installed Plug-ins

To find out whether a plugin is included in your JavaScript file and ready for use, look in the **Plugins Section** of the JavaScript file. The following example shows what the `getQueryParam` function looks like in the **Plugins Section**.

```

/***** PLUGINS SECTION *****/

/* You may insert any plugins you wish to use here. */
/*
 * Plugin: getQueryParam 1.3 - Return query string parameter values
 */
s.getQueryParam=new Function("qp","d","
+\"vars=this,v='',i,t;d=d?d:'';while(qp){i=qp.indexOf(',');i=i<0?qp.l\"
//
// ... more code below ...
//

```

getAndPersistValue

The `getAndPersistValue` plug-in obtains a value of your choosing and populates it into a Analytics variable for a determined period. A common use is to see how many page views a campaign generates after a click-through, which enables you to easily see the most common pages for each campaign.



Important: This plug-in has not been validated to be compatible with [AppMeasurement for JavaScript](#). See [AppMeasurement Plug-in Support](#).

For example, you might use this plug-in to set a campaign tracking code from the `campaign` variable into a Custom Traffic (`s.prop`) variable on each visitor's page view made for the next 30 days. This example lets you determine how many page views the tracking code generated as a result of the original click-through.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Plug-in Code and Implementation

CONFIG SECTION: No changes required for this section.

Plug-in Configuration

Place the following code within the `s_doPlugins()` function, which is located in the area of the `s_code.js` file labeled *Plugin Config*. Choose one Custom Traffic (`s.prop`) variable or one Custom Conversion (`s.eVar`) variable for use in capturing persisted value data. This should be a variable that has been enabled using the Admin Console, but which is not currently in use for any other purpose. You can use the following example and update it based on your requirements.

```
s.prop1=s.getAndPersistValue(s.campaign,'s_getval',30);
```

`s.getAndPersistValue` has three arguments:

1. Currently populated variable or value to persist (`s.campaign` shown above).
2. Cookie name, used to store the value (`s_getval` shown above).
3. Period of time for persistence, in days. "30" as shown above would cause the value to be populated into the selected variable on every page view made by the user for the next 30 days. If omitted, the setting defaults to *session*.

PLUGINS SECTION: Add the following code to the area of the `s_code.js` file labeled PLUGINS SECTION. Do not make any changes to this portion of the plug-in code.

```
/*
 * Plugin: getAndPersistValue 0.3 - get a value on every page
 */
s.getAndPersistValue=new Function("v","c","e",""
+"var s=this,a=new Date;e=e?e:0;a.setTime(a.getTime()+e*86400000);if("
+"v)s.c_w(c,v,e?a:0);return s.c_r(c);");
```

Always test plug-in installations extensively to ensure that data collection is as expected before deploying in a production environment.

getDaysSinceLastVisit

Determines the number of days since a user last visited your site, and captures this information in a Analytics variable.



Important: [Analysis Workspace](#) now includes a **Days since last visit** dimension out of the box, thereby nullifying the need for this plugin.

This return frequency data can be used to answer the following questions:

- How frequently do users revisit my site?
- How does return frequency correlate with conversion? Do repeat buyers visit frequently or infrequently?
- Do users who click through my campaigns then return frequently?

The plug-in can also generate values used for segmentation. For example, you can create a segment to view all of the data for only those visits that were preceded by 30 or more days of non-visitation by the user.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Plug-in Code and Implementation

CONFIG SECTION

No changes required.

Plugin Config

Place the following code within the `s_doPlugins()` function, which is located in the area of the `s_code.js` file labeled *Plugin Config*. Choose one Custom Traffic (`s.prop`) variable and/or one Custom Conversion (`s.eVar`) variable for use in capturing return frequency data. This selection should be a variable that has been enabled using the Admin Console but is not currently in use for any other purpose. The following is given as an example, and should be updated appropriately based on your needs.

```
s.prop1=s.getDaysSinceLastVisit(Cookie_Name);
```

PLUGINS SECTION

Add the following code to the area of the `s_code.js` file labeled *PLUGINS SECTION*. Do not make any changes to this portion of the plug-in code.

```
/*
 * Plugin: Days since last Visit 1.1 - capture time from last visit
 */
s.getDaysSinceLastVisit=new Function("c",""
+"var s=this,e=new Date(),es=new Date(),cval,cval_s,cval_ss,ct=e.getT"
+"ime(),day=24*60*60*1000,f1,f2,f3,f4,f5;e.setTime(ct+3*365*day);es.s"
+"etTime(ct+30*60*1000);f0='Cookies Not Supported';f1='First Visit';f"
```

```

+ "2='More than 30 days';f3='More than 7 days';f4='Less than 7 days';f"
+ "5='Less than 1 day';cval=s.c_r(c);if(cval.length==0){s.c_w(c,ct,e);"
+ "s.c_w(c+'_s',f1,es);}else{var d=ct-cval;if(d>30*60*1000){if(d>30*da"
+ "y){s.c_w(c,ct,e);s.c_w(c+'_s',f2,es);}else if(d<30*day+1 && d>7*day"
+ "){s.c_w(c,ct,e);s.c_w(c+'_s',f3,es);}else if(d<7*day+1 && d>day){s."
+ "c_w(c,ct,e);s.c_w(c+'_s',f4,es);}else if(d<day+1){s.c_w(c,ct,e);s.c"
+ "_w(c+'_s',f5,es);}}else{s.c_w(c,ct,e);cval_ss=s.c_r(c+'_s');s.c_w(c"
+ "+'_s',cval_ss,es);}}cval_s=s.c_r(c+'_s');if(cval_s.length==0) retur"
+ "n f0;else if(cval_s!=f1&& cval_s!=f2&& cval_s!=f3&& cval_s!=f4&& cval_s"
+ "!=f5) return '';else return cval_s;");

```

Notes

- Always test plug-in installations extensively to ensure that data collection is as expected before deploying in a production environment.
- The plug-in categorizes users by return frequency into the following groups:
 - First Visit
 - Less than 1 day
 - Less than 7 days
 - More than 7 days
 - More than 30 days
- This plug-in relies on cookies to flag a user has having visited previously. If the browser does not accept cookies, the plug-in returns a value of *Cookies Not Supported*.

getLoadTime

Gets the page load time in tenths of a second and lets you store the value in a prop, eVar, and/or a numeric event.

To use this plugin, you insert the function code, then call the function twice in your `s_code.js` file. Once at the beginning of the file, and then again in the `doPlugins` section. This plugin is intentionally not defined as a method of the `s` object. Doing so would have added to the calculated page load time.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Plug-in Code and Implementation

Add the function

Add the following definition of the `s_getLoadTime` function in `s_code.js`, anywhere before the "DO NOT ALTER ANYTHING BELOW THIS LINE" section:

```

function s_getLoadTime(){if(!window.s_loadT){var b=new
Date().getTime(),a=window.performance?performance.timing:0,a=a?o.requestStart:window.inHeadIS||0;s_loadT=a?Math.round((b-a)/100):''}return
s_loadT}

```

Make the initial function call

Add a call to `s_getLoadTime()` near the beginning of `s_code.js`, outside of any function.

Make the final function call

Add another call to `s_getLoadTime()` in the `s_doPlugins()` function, saving the returned value in a prop, eVar, and/or a numeric event.

Usage Example 1 - Save the page load time in prop10 and eVar20:

```

s.eVar20=s.prop10=s_getLoadTime();

```

Usage Example 2 - Save the page load time in numeric event99:

```
if(s_getLoadTime())s.events=s.apl(s.events,'event90='+s_getLoadTime(),',',1);
```

(Optional) Add support for older browsers

To support older browsers that don't provide the [window.performance.timing](#) property, include the following line in the HEAD section of the page's HTML near the beginning and prior to invoking .js, .css, or other files:

```
<script type="text/javascript">var inHeadTS=(new Date()).getTime();</script>
```

getNewRepeat

Determines whether a visitor is a new visitor or a repeat visitor, and captures this information in a Analytics variable.

Use this plug-in to answer the following questions:

- What percentage of my visitors are new (as opposed to repeat) visitors?
- Do return visitors generate higher conversion per capita than new visitors? What is this ratio?
- Do my marketing campaigns cause persistence across visits? For example, do users who click through my campaigns then return later?



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Plug-in Code and Implementation

CONFIG SECTION: No changes required for this section.

Plug-in Configuration

Place the following code within the `s_doPlugins()` function, which is located in the area of the `s_code.js` file labeled *Plugin Config*. Choose one Custom Traffic (`s.prop`) variable or one Custom Conversion (`s.eVar`) variable for use in capturing persisted value data. This should be a variable that has been enabled using the Admin Console, but which is not currently in use for any other purpose. You can use the following example and update it based on your requirements.

```
s.prop1=s.getNewRepeat(30,'s_getNewRepeat');
```

`s.getNewRepeat` has two optional arguments:

1. The first optional argument is the number of days that the cookie should last. The default value if this argument is omitted is 30 days.
2. The second optional argument is the cookie name. A default value is used if this argument is omitted.

PLUGINS SECTION: Add the following code to the area of the `s_code.js` file labeled PLUGINS SECTION. Do not make any changes to this portion of the plug-in code.

```
/*
 * Plugin: getNewRepeat 1.2 - Returns whether user is new or repeat
 */
s.getNewRepeat=new Function("d","cn",""
+"var s=this,e=new Date(),cval,sval,ct=e.getTime();d=d?d:30;cn=cn?cn:"
+" 's_nr' ;e.setTime(ct+d*24*60*60*1000);cval=s.c_r(cn);if(cval.length="
+"=0){s.c_w(cn,ct+'-New',e);return'New';}sval=s.split(cval,'-');if(ct"
+"-sval[0]<30*60*1000&&sval[1]=='New'){s.c_w(cn,ct+'-New',e);return'N"
+"ew';}else{s.c_w(cn,ct+'-Repeat',e);return'Repeat';}");
/*
 * Utility Function: split v1.5 (JS 1.0 compatible)
 */
```




```
s.split=new Function("l","d",""
+"var i,x=0,a=new Array;while(l){i=l.indexOf(d);i=i>-1?i:l.length;a[x"
+"++] =l.substring(0,i);l=l.substring(i+d.length);}return a");
```

Always test plug-in installations extensively to ensure that data collection is as expected before deploying in a production environment.

getPageVisibility


Records the number of seconds your page was the active tab within the browser, and passes that value into a metric on the next page view.

 **Note:** This is a beta version of the plugin, and additional updates may be forthcoming.

This plug-in requires [getVisitStart](#).

This plug-in also records the total seconds the page was within the browser (both active and passive viewing time). It is required to use the [getPreviousValue](#) plug-in in order to track the previous page name associated with the page visibility events. Tracking these values helps you better understand visitor engagement and more accurately track visitor behavior on your sites.

It is required that you use the [getPreviousValue](#) plug-in to track the previous page name associated with the page visibility events. Tracking these values helps you better understand visitor engagement and more accurately track visitor behavior on your sites.

 **Note:** The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics. This plug-in is compatible only with [AppMeasurement](#) tracking libraries.

Required Supporting Plug-ins

- [appendList](#)
- [getPreviousValue](#)
- [getVisitStart](#)

Plug-in Code and Implementation

Config Section

The `s.pvel` variable should contain the three events you wish to use:

Event	Definition
Total Page Visibility Seconds (Numeric)	The amount of time the page was active within the browser
Total Page Seconds (Numeric)	The amount of time the page was loaded in the browser, regardless of its visibility state
Total Page Visibility Instances (Counter)	The total number of times a value was recorded for the previous two events

Sample Calls

```
//Page Visibility Event List (total page visibility seconds, total page seconds, total page
visibility instances)
s.pvel='event7,event8,event9'
```

doPlugins Section

To initialize the plug-in, two lines of code are required in the `doPlugins` section of your `s_code`, preferably after you have designated the `s.pageName` variable.

Sample Calls

```
/* Page Visibility */
s.eVar9 = s.getPreviousValue(s.pageName,'gpv_v9',''); //Record the previous page name in the
designated eVar of your choice
s.getPageVisibility();
```

Plug-ins Section

```
/* Page Visibility Plugin 0.1 (BETA) */
s.getPageVisibility=new Function("", ""
+"var s=this;if(s.getVisitStart()){s.Util.cookieWrite('s_pvs','');s.U"
+"til.cookieWrite('s_tps','');if(s.Util.cookieRead('s_pvs')&&s.pvt<1"
+""){if(parseInt(s.Util.cookieRead('s_pvs'))<=parseInt(s.Util.cookieRe"
+"ad('s_tps'))){s.pve=s.pvel.split(',')&s.events=s.apl(s.events,s.pve"
+"[0]+''+(parseInt(s.Util.cookieRead('s_pvs'))),',',2);s.Util.cookie"
+"Write('s_pvs','');s.events=s.apl(s.events,s.pve[1]+''+(parseInt(s."
+"Util.cookieRead('s_tps'))),',',2);s.Util.cookieWrite('s_tps','');s."
+"events=s.apl(s.events,s.pve[2],',',2);}s.pvi=setInterval(s.pvx,100"
+"0);s.wpvi=setInterval(s.wpvc,5000);"};
s.gbp=new Function("", ""
+"if('hidden'in document){return null;}var bp=['moz','ms','o','webkit"
+"'];for(var i=0;i<bp.length;i++){var p=bp[i]+'Hidden';if(p in docume"
+"nt){return bp[i];}}return null;");
s.hp=new Function("p", ""
+"if(p){return p+'Hidden';}else{return'hidden';}");
s.vs=new Function("p", ""
+"if(p){return p+'VisibilityState';}else{return'visibilityState';}");
s.ve=new Function("p", ""
+"if(p){return p+'visibilitychange';}else{return'visibilitychange';}");
s.pvx=new Function("", ""
+"s.pvt+=1;");
s.wpvc = function(){var tempDate = Date.now();s.Util.cookieWrite('s_tps',
Math.ceil((tempDate - s.totalTime)/1000));s.Util.cookieWrite('s_pvs', s.pvt)}
document.addEventListener('visibilitychange',function(event){if(document.hidden){s.visibility
=
false;clearTimeout(s.pvi);}else{s.visibility=true;s.pvi=setInterval(s.pvx,1000);}});s.totalTime=new
Date();s.pvt=0;s.prefix=s.gbp;s.hidden=s.hp(s.prefix);s.visibility=true;s.visibilityState=s.vs(s.prefix);s.visibilityEvent=s.ve(s.prefix);
```

Notes

- Always test plug-in installations to ensure that data collection is as expected before deploying in a production environment.
- Because the plug-in passes the page visibility seconds and total seconds as they are associated with the previous page, data is not collected for the final page view of the visit.
- This plug-in relies on the ability to set cookies in the user's web browser. If the user does not accept first-party cookies, the plug-in will not pass data into Analytics.
- The plug-in creates its own first-party cookies named `s_tps` and `s_pvs`.
- A very small percentage of users will not pass percentage of page viewed data due to browser limitations, and logic is contained within the plugin to ensure that the data is not skewed as a result. However, this plug-in has been successfully tested in IE, Firefox, Chrome, and Safari.

- Due to the way the plugin measures total seconds and associates that value with the previous page name, there will be differences between default time spent on page metrics and total seconds metrics.
- Calculated metrics can be created to aid in summarizing and understanding visitor behavior associated with these metrics:
 - **Page Visibility Ratio** (Total Page Visibility Seconds / Total Page Seconds)
 - **Total Hidden Seconds** (Total Page Seconds – Total Page Visibility Seconds)
 - **Average Page Visibility Seconds** (Total Page Visibility Seconds/Total Page Visibility Instances)
 - **Average Page Hidden Seconds** ((Total Page Seconds - Total Page Visibility Seconds)/Total Page Visibility Instances)
- Due to the way the plug-in rounds up the seconds, there can be a 1-2 second difference between the total page visibility seconds and total seconds, with total seconds being higher. (To be resolved in a future update)
- Using the `getVisitStart` plug-in should account for visitors that have a new visit start after a period of 30+ minutes of inactivity. This is not working as designed; however, there will likely be a workaround when we incorporate the “total active seconds” in a future iteration of the plug-in.

Frequently Asked Questions

Will this plug-in make additional server calls?

The plugin will only record page visibility values on subsequent page view server calls. No additional server calls are used in conjunction with it.

If I do not want to capture total page seconds or total page visibility instances, can I leave those out of the event list?

Yes, the total page seconds and total visibility instances are optional events and can be left out of the list if desired.

Will the events captured make sense if I use them in reports other than the Previous Page Name?

Since the plugin records values on the subsequent image request only other eVars that have been captured in a ‘previous page’ context could be applied, i.e. ‘Previous Page URL’.

Will the plug-in send the visibility time on an `s.tl()` call, or only on an `s.t()` call?

The visibility time is only recorded with `s.t()` calls.

`getVisitStart`

Determines if a new visit is starting.

Configuration Variables

None

Parameters

`c` = (string) cookie name for tracking.

Returns

(integer) 1 on first page of visit, otherwise 0.

Sample Calls

```
s.eVar50 = s.getVisitStart("s_visit");
```

getPercentPageViewed

The **getPercentPageViewed** plugin measures a visitor's scroll activity to see how much of a page the visitor viewed before moving on to another page.



Note: You do not need to use the **getPercentPageViewed** plugin if your web pages are small in height and there is no need to measure how far visitors scroll down. Also, if you want to measure scroll activity on exit pages only, you cannot use this plugin.

Prerequisites

You must have **AppMeasurement** and the [handlePPVevents](#) helper plugin to run the **getPercentPageViewed** plugin.

Implementation

To implement this plugin, copy and paste the code to anywhere within the Plugins section of the AppMeasurement file.



Note: Adding the **bolded** comments/version numbers of the [code](#) to the AppMeasurement file helps Adobe Consulting with troubleshooting any potential implementation issues.

You can run the **getPercentPageViewed** function as needed within the doPlugins function (see example calls below.)

Arguments to pass in

pid (optional, string)	A page identifier that is correlated with the percentages provided by the plugin measurements. It defaults to the Analytics pageName variable or the URL if the pageName variable is not set.
ch (optional, Boolean)	"True" is the recommended/default value for this argument. Set this equal to "false" if you do not want this plugin to consider any changes made to a page's size after its initial load, due to SPA code, dynamic HTML, etc.

Returns

The **getPercentPageViewed** plugin returns nothing. Instead, it sets the following variables within the AppMeasurement object:

- **s_ppvPreviousPage:** The name of the previous page viewed (because final measurements are not available until an new page loads.)
- **s_ppvHighestPercentViewed:** The highest percent of the previous page that the visitor viewed (height-wise). In other words, the furthest point that the visitor scrolled down on the previous page.
- **s_ppvInitialPercentViewed:** The percentage of the previous page that was visible when the previous page **first** loaded.
- **s_ppvHighestPixelSeen:** The highest number of total pixels seen (height-wise) as the visitor scrolled down the previous page.

Cookies

The **getPercentPageViewed** plugin creates a cookie, called s_ppv, that is passed from page to page. The contents of the cookie contain the values inserted in the four variables described above and expire at the end of the session.

Example Calls

Sample Call 1

```
if(s.pageName) s.getPercentPageViewed();
if(s._ppvPreviousPage)
{
s.prop1 = s._ppvPreviousPage;
s.prop2 = "highestPercentViewed=" + s._ppvHighestPercentViewed + " | initialPercentViewed=" +
s._ppvInitialPercentViewed;
}
```

The code sample above:

- Determines if s.pageName is set and if so, the code will run the getPercentPageViewed function
- When the getPercentPageViewed function runs, it creates the variables described in the "Returns" section above.
- If the "Returns" variables were successfully set:
 - The code sets s.prop1 equal to the value of s._ppvPreviousPage (i.e. the previous value of s.pageName, or the previous page.)
 - The code also sets s.prop2 equal to the Highest Percentage Viewed of the previous page and the Initial Percentage Viewed of the previous page.



Note: If an entire page is visible when it first loads, both the Highest Percentage Viewed and the Initial Percentage Viewed dimensions would be equal to 100. However, if an entire page is not visible when it first loads, but the visitor never scrolls down the page before moving on to the next page, then both the Highest Percentage Viewed and the Initial Percentage Viewed dimensions would be equal to the same value.

Sample Call 2

Assume that s.prop5 has been set aside to capture a rolled-up "page type" rather than the entire page name.

The following code determines if s.prop5 has been set and, if so, stores its value as the "previous page" to correlate with the Highest Percentage Viewed and the Initial Percentage Viewed dimensions. The value is still stored in the s._ppvPreviousPage variable but can be treated as if it were the previous page type instead of the previous page name.

```
if(s._ppvPreviousPage)
{
s.prop1 = s._ppvPreviousPage;
s.prop2 = "highestPercentViewed = " + s._ppvHighestPercentViewed + " | initialPercentViewed="
+ s._ppvInitialPercentViewed;
}
```

S Object Replacement

When instantiating the main AppMeasurement library object with a name other than "s", change the following portion of the plugin code from this:

```
s.getPercentPageViewed=function(pid,ch)
```

to this:

```
[objectname].getPercentPageViewed=function(pid,ch)
```

Code to Deploy

Plugins Section: Add the following code to the area of the s_code.js file labeled PLUGINS SECTION. Do not make any changes to this portion of the plug-in code.

```
/* ***** BEGIN CODE TO DEPLOY *****
/* Adobe Consulting Plugin: getPercentPageViewed v3.01 w/handlePPVevents helper function
```

```

(Requires AppMeasurement and p_fo plugin) */
s.getPercentPageViewed=function(pid,ch){var
s=this,a=s.c_r("s_ppv");a=-1<a.indexOf(",")?a.split(","):[];a[0]=s.unescape(a[0]);
pid=pid?pid:s.pageName?s.pageName:document.location.href;s.ppvChange=ch?ch:!0;if("undefined"===typeof
s.linkType||"o"!==s.linkType)s.ppvID&&s.ppvID===pid||(s.ppvID=pid,s.c_w("s_ppv",""),s.handlePPVEvents()),s.p_fo("s_gppvLoad")&&window
.addEventListener&&(window.addEventListener("load",s.handlePPVEvents,!1),window.addEventListener("click",s.handlePPVEvents,
!1),window.addEventListener("scroll",s.handlePPVEvents,!1),window.addEventListener("resize",s.handlePPVEvents,!1)),s.ppvPreviousPage
=a[0]?a[0]:"",s.ppvHighestPercentViewed=a[1]?a[1]:"",s.ppvInitialPercentViewed=a[2]?a[2]:"",s.ppvHighestPixelsSeen=a[3]?a[3]:"";

/* Adobe Consulting Plugin: handlePPVEvents helper function (for getPercentPageViewed v3.01
Plugin) */
s.handlePPVEvents=function(){if("undefined"!==typeof s_c_il){for(var
c=0,d=s_c_il.length;c<d;c++)if(s_c_il[c]&&
s_c_il[c].getPercentPageViewed){var a=s_c_il[c];break}if(a&&a.ppvID){var
f=Math.max(Math.max(document.body.scrollHeight,
document.documentElement.scrollHeight),Math.max(document.body.offsetHeight,document.documentElement.offsetHeight),Math.max(document.
body.clientHeight,document.documentElement.clientHeight));var g=f?Math.min(Math.max(
(c/f*100),100),100):var
e="";!a.c_r("s_tp")||a.unescape(a.c_r("s_ppv").split(",")[0])!==a.ppvID||!1===a.ppvChange&&
a.c_r("s_tp")&&f!=a.c_r("s_tp")?(a.c_w("s_tp",f),a.c_w("s_ppv","")):e=a.c_r("s_ppv");var
b=e&&-1<e.indexOf(",")?e.split(",",4):[];f=0<b.length?b[0]:
escape(a.ppvID);var
g=1<b.length?parseInt(b[1]):d,h=2<b.length?parseInt(b[2]):d;b=3<b.length?parseInt(b[3]):c;0<d&&(e=f+"",
+(d>g?d:g)+"", "+h+", "+(c>b?c:b));a.c_w("s_ppv",e)}}};

/* Adobe Consulting Plugin: p_fo (pageFirstOnly) v2.0 (Requires AppMeasurement) */
s.p_fo=function(on){var
s=this;s.__fo||(s.__fo={});if(s.__fo[on])return!1;s.__fo[on]={};return!0};
/***** END CODE TO DEPLOY *****/
}

```

getPreviousValue

Captures the value of a Analytics variable on the next page view. For example, you can use plug-in to capture the `s.pageName` value from the previous page view into a Custom Traffic variable. It also has an option to capture a previous value only when designated success events are set.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Plug-in Code and Implementation

CONFIG SECTION: No changes required for this section.

Plug-in Configuration

Place the following code within the `s.doPlugins()` function, which is located in the area of the `s_code.js` file labeled *Plugin Config*. Choose one Custom Traffic (`s.prop`) variable or one Custom Conversion (`s.eVar`) variable for use in capturing persisted value data. This should be a variable that has been enabled using the Admin Console, but which is not currently in use for any other purpose. You can use the following example and update it based on your requirements.

```
s.prop1=s.getPreviousValue(s.pageName,'gppv_pn','event1');
```

`s.getPreviousValue` has three arguments:

1. The variable to be captured from the previous page (`s.pageName` above).

2. The cookie name for use in storing the value for retrieval (*gpv_pn* above).
3. The events that must be set on the page view in order to trigger the retrieval of the previous value (*event1* above). When left blank or omitted, the plug-in captures the previous value on all page views.

PLUGINS SECTION: Add the following code to the area of the `s_code.js` file labeled PLUGINS SECTION. Do not make any changes to this portion of the plug-in code.

```

/*
 * Plugin: getPreviousValue_v1.0 - return previous value of designated
 * variable (requires split utility)
 */
s.getPreviousValue=new Function("v","c","el",""
+"var s=this,t=new Date,i,j,r='';t.setTime(t.getTime()+1800000);if(el"
+"){if(s.events){i=s.split(el,',');j=s.split(s.events,',');for(x in i"
+"){for(y in j){if(i[x]==j[y]){if(s.c_r(c)) r=s.c_r(c);v?s.c_w(c,v,t)"
+":s.c_w(c,'no value',t);return r}}}}else{if(s.c_r(c)) r=s.c_r(c);v?"
+s.c_w(c,v,t):s.c_w(c,'no value',t);return r}");
/*
 * Utility Function: split v1.5 - split a string (JS 1.0 compatible)
 */
s.split=new Function("l","d",""
+"var i,x=0,a=new Array;while(l){i=l.indexOf(d);i=i>-1?i:l.length;a[x"
+"++]=l.substring(0,i);l=l.substring(i+d.length);}return a");

```

Notes

- Always test plug-in installations extensively to ensure that data collection is as expected before deploying in a production environment.
- If no value is present for the selected variable on any given page, the text *no value* will be set in the cookie.
- A fixed 30-minute cookie expiration is now set for each cookie, and refreshed with each page load. The plug-in works for the length of a visit.
- Because the function must be called as part of the plug-ins section of code, the code runs each time *s.t()* or *s.tl()* is called.
- The chosen variable should be populated with a value prior to the call to *s.getPreviousValue*. Because the *s_doPlugins()* function is executed after the variables on the page are populated, this issue rarely occurs. It should only be a matter of concern if the variable used with this plug-in is populated within the *s_doPlugins()* function and after the call to *s.getPreviousValue*.

getQueryParam

Returns the value of a specified query string parameter, if found in the current page URL. Because important data (such as campaign tracking codes, internal search keywords, etc.) is available in the query string on a page, *getQueryParam* helps capture the data into Analytics variables.



Important: This plug-in is used by H code only. [AppMeasurement for JavaScript](#) provides this functionality natively using [Util.getQueryParam](#).

Once installed in your AppMeasurement for JavaScript code, the plug-in is configured by selecting a Analytics variable to populate using data found in the query string, and specifying which query string values to capture. The plug-in detects the specified query string, if present, and populates the chosen variable with its value. If no query string parameter is found with that value, an empty string is returned. If a query string parameter exists but does not have a value (such as `param1` in `?param1¶m2=value`), the word *true* is returned.



Note: The base code for the plug-in must be installed in your AppMeasurement for JavaScript code before the examples below will work.

If you wanted to use `s.campaign` to capture campaign tracking codes available as values of the `cid` query parameter, you would enter the following in the `doPlugins()` function in your AppMeasurement for JavaScript code:

```
s.campaign=s.getQueryParam('cid')
```

In this example, if the user arrived at a landing page on your site where the URL was `http://www.yoursite.com/index.html?cid=123456`, then `s.campaign` would receive a value of `123456`. This could be seen using the DigitalPulse Debugger, which should show `v0=123456` as part of the image request.



Note: The parameter `cid` and others are used here as examples. You can replace them with any query string parameters that exist on your site.

The `getQueryParam` plug-in has two additional arguments (options) that can be used to capture data into Analytics variables:

```
s.getQueryParam('p','d','u')
```

where:

```
p = comma-separated list of query parameters to locate (can also be a single value with no comma)
d = delimiter for list of values (in case more than one specified parameter is found)
u = where to search for value (e.g., document.referrer); set to current page URL by default
```

If `p` is a list of query string parameters and more than one query string parameter is found in the URL, all values are returned in a list separated by the delimiter, `d`, which can be a single character or a string of characters, such as " : " (space-colon-space). If `d` is omitted, no delimiter is used between values. If one query string parameter should take precedence over another, when both are found, use an `if` statement as shown below.

```
// cid takes precedence over iid if both exist in the query string
s.campaign=s.getQueryParam('cid');
if(!s.campaign)
s.campaign=s.getQueryParam('iid');
```

As of version `getQueryParam` v2.0, the plug-in accepts an optional third argument, `u`, which allows you to specify the URL from which you would like to extract query string parameters. By default (i.e. if this third argument is omitted or left blank), the plug-in uses the page URL. For example, if you would like extract a query string from the referrer, you can use the following code:

```
// take the query string from the referrer
s.eVar1=s.getQueryParam('pid','','document.referrer');
```

The flag "f" should be used in this third argument with frames, when the necessary query string parameter is found in the address bar rather than the current frame's URL:

```
// take the query string from the parent frame
s.eVar1=s.getQueryParam('pid','','f');
```

When using frames and the `f` parameter, it is recommended that the `getValOnce` plug-in be used to prevent the campaign tracking code to be sent with each page view.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Plug-in Code

```
/*
 *
 * Main Plug-in code (should be in Plug-ins section)
 *
 */
```



```

* Plugin: getQueryParam 2.3
*/
s.getQueryParam=new Function("p","d","u",""
+"var s=this,v='',i,t;d=d?d:'';u=u?u:(s.pageURL?s.pageURL:s.wd.locati
+"on);if(u=='f')u=s.gtfs().location;while(p){i=p.indexOf(',');i=i<0?p
+ ".length;i;t=s.p_gpv(p.substring(0,i),u+'');if(t){t=t.indexOf('#')>-
+"1?t.substring(0,t.indexOf('#')):t;}if(t)v+=v?d+t:t;p=p.substring(i="
+"=p.length?i:i+1)}return v");
s.p_gpv=new Function("k","u",""
+"var s=this,v='',i=u.indexOf('?'),q;if(k&&i>-1){q=u.substring(i+1);v
+"=s.pt(q,'&','p_gvf',k)}return v");
s.p_gvf=new Function("t","k",""
+"if(t){var s=this,i=t.indexOf('='),p=i<0?t:t.substring(0,i),v=i<0?'T
+"rue':t.substring(i+1);if(p.toLowerCase()==k.toLowerCase())return s."
+"epa(v)}return ''");

/*****
*
* Commented example of how to use this is doPlugins function
*
*****/
/* Plugin Example: getQueryParam 2.3
//single parameter
s.campaign=s.getQueryParam('cid');

//multiple parameters
s.campaign=s.getQueryParam('cid,sid','');

//non-page URL example
s.campaign=s.getQueryParam('cid','','document.referrer);

//parent frame example
s.campaign=s.getQueryParam('cid','','f');

*/

/*****
*
* Config variables (should be above doPlugins section)
*
*****/

None

/*****
*
* Utility functions that may be shared between plug-ins (name only)
*
*****/

None

```

getTimeParting

The *getTimeParting* plug-in populates custom variables with hour of day, day of week, and weekend and weekday values into custom variables. Analysis Workspace offers out-of-the-box [Time Parting dimensions](#). The plug-in should be used if time parting dimensions are needed in other Analytics solutions, outside of Analysis Workspace.

This plug-in captures the date and time information available in the user's web browser. It obtains the hour of the day and the day of the week from this information. It then converts this data to the time zone of your choosing. It also accounts for Daylight Savings Time.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Plug-in Code

Config Section

Place the following code in the area of the `s_code.js` file labeled **CONFIG SECTION**, and make the necessary updates as described below.

`s._tpDST` - an array of DST values. The array is structured in the following format: `YYYY: 'MM/DD,MM/DD'`

```
//time parting configuration
//Australia
s._tpDST = {
2012: '4/1,10/7',
2013: '4/7,10/6',
2014: '4/6,10/5',
2015: '4/5,10/4',
2016: '4/3,10/2',
2017: '4/2,10/1',
2018: '4/1,10/7',
2019: '4/7,10/6' }

//US
s._tpDST = {
2012: '3/11,11/4',
2013: '3/10,11/3',
2014: '3/9,11/2',
2015: '3/8,11/1',
2016: '3/13,11/6',
2017: '3/12,11/5',
2018: '3/11,11/4',
2019: '3/10,11/3' }

//Europe
s._tpDST = {
2012: '3/25,10/28',
2013: '3/31,10/27',
2014: '3/30,10/26',
2015: '3/29,10/25',
2016: '3/27,10/30',
2017: '3/26,10/29',
2018: '3/25,10/28',
2019: '3/31,10/27' }
```

Note for Northern Hemisphere clients: in the array DST values are DST start, DST end.

Note for Southern Hemisphere clients: in the array DST values are DST end, DST start.

Parameters

```
var tp = s.getTimeParting(h,z);
```

- **h** = (required) Hemisphere - Specify what hemisphere you are converting the time to. This is a value of 'n' or 's'. This is used to determine how to use the DST array passed. If 'n' is passed the plugin uses the dates when DST is on. If 's' is passed the plugin uses the dates when DST is off.
- **z** = (optional) Time Zone - If you would like the data to be based upon a specific time period, then that will need to be specified as the hours different from GMT here. Note this should be the GMT during non DST. If no value is specified, it defaults to GMT (i.e. '-5' for US Eastern Time)

Returns

Returns a concatenated value of time at minute level and day of week, for example:

```
8:03 AM|Monday
```

You can then use [Classifications](#) to group visits into time periods. For example, you could set up a rule in Classification Rule Builder to bucket visits between 9:00 AM and 9:59 AM to "9:00 AM - 10:00 AM". As an alternative to classifications, you could provide additional client-side logic to bucket visits in JavaScript.

Example Call

```
var tp = s.getTimeParting('n','-7');
s.prop1 = tp;
```

PLUGINS SECTION

Add the following code to the **PLUGINS SECTION** in the `s_code.js` file.

```
/*
 * Plugin: getTimeParting 3.4
 */
s.getTimeParting=new Function("h","z",""
+"var s=this,od;od=new Date('1/1/2000');if(od.getDay()!6||od.getMont"
+"h()!0){return'Data Not Available';}else{var H,M,D,U,ds,de,tm,da=['"
+"Sunday','Monday','Tuesday','Wednesday','Thursday','Friday','Saturda"
+"y'],d=new Date();z=z?z:0;z=parseFloat(z);if(s._tpDST){var dso=s._tp"
+"DST[d.getFullYear()].split(/,/);ds=new Date(dso[0]+'/'+'d.getFullYea"
+"r());de=new Date(dso[1]+'/'+'d.getFullYear());if(h=='n'&&d>ds&&d<de)"
+"{z=z+1;}else if(h=='s'&&(d>de||d<ds)){z=z+1;}}d=d.getTime()+(d.getT"
+"imezoneOffset()*60000);d=new Date(d+(3600000*z));H=d.getHours();M=d"
+".getMinutes();M=(M<10)?'0'+M:M;D=d.getDay();U=' AM';if(H>=12){U=' P'"
+"M';H=H-12;}if(H==0){H=12;}D=da[D];tm=H+' '+M+U;return(tm+' '+D);}");
```

Notes

- Always test plug-in installations to ensure that data collection is as expected before deploying to production.
- Configuration variables must be set for plugin to work correctly.

getValOnce

The `getValOnce` plug-in prevents a given variable from being set to the previously defined value. It uses a cookie to determine a variable's last seen value. If the current value matches the cookie value, the variable is overwritten with a blank string before it is sent to Adobe's processing servers. This plug-in is useful to prevent conversion variable instance inflation caused when users refresh the page or click the Back button.



Important: This plug-in has not been validated to be compatible with [AppMeasurement for JavaScript](#). See [AppMeasurement Plug-in Support](#).

Parameters

```
s.eVar1=s.getValOnce(variable,cookie,expiration,minute);
```

- **Variable:** The variable that will be checked. This is typically the same as the variable being defined.
- **Cookie:** The name of the cookie that stores the previous value to compare against. The cookie can be any value.
- (Optional) **Expiration:** The number of days the cookie will expire. If not set or set to 0, the default expiration is the browser session.
- (Optional) **Minute:** If you set this to the string value *m*, the expiration value is defined in minutes instead of days. If not set, *days* is the default expiration.

Properties

- This plug-in is commonly used on conversion variables. However, you can use it on any Analytics variable.
- When Javascript encounters this function, it compares the defined value to what is stored in the cookie. If the defined value is different from the cookie value, the defined value is set. If the defined value is the same as the cookie value, an empty string is returned.
- The cookie can only store a single value, meaning the plug-in only looks as the last defined value.
- The plug-in does not stop all values from defining the variable after it is defined. The plug-in only prevents the last value from being set multiple times consecutively.
- If the end user blocks or rejects cookies, the original value is always returned.

- The plug-in's session is different from what Analytics defines as a session (or visit). Analytics terminates a session after 12 hours of activity or 30 minutes of inactivity. Because the plug-in uses the browser's session definition, it is terminated only after the user closes the tab or exits the browser.
- If a user closes your page, opens a different tab and navigates back to your site within 30 minutes, the plug-in creates a new session while keeping the Analytics visit open.
- If a user keeps the browser window open without clicking on a link for more than 30 minutes, the Analytics visit expires while keeping the browser session open.



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

Implementation



Note: If your organization uses Marketing Channels and has rules set up based on `s.campaign`, it is recommended that you not use the `getValOnce` plugin when setting the `s.campaign` value. Doing so could lead to an incorrect channel being assigned on a secondary campaign click-through.

To implement this plug-in, place the following code within your `s_code.js` file

```

/*****
 *
 * Main Plug-in code (should be in Plug-ins section)
 *
 *****/
/*
 * Plugin: getValOnce_v1.11
 */
s.getValOnce=new Function("v","c","e","t",""
+"var s=this,a=new Date,v=v?v:'' ,c=c?c:'s_gvo',e=e?e:0,i=t=='m'?6000"
+"0:86400000,k=s.c_r(c);if(v){a.setTime(a.getTime()+e*i);s.c_w(c,v,e"
+"==0?0:a);}return v==k?'':v");

```

Once the above code is implemented, define the desired variable using the `getValOnce` function. The following are several examples on how it can be implemented:

Preventing the same campaign value from being defined if a duplicate value is detected within 30 days of cookie being set:

```
s.campaign=s.getValOnce(s.campaign,'s_cmp',30);
```

Prevents the same eVar1 value from being defined if a duplicate value is detected within 30 minutes of the cookie being set:

```
s.eVar1=s.getValOnce(s.eVar1,'s_ev1',30,'m');
```

Prevents the same eVar2 value from being defined multiple times in the same browser session:

```
s.eVar2=s.getValOnce(s.eVar2,'s_ev2');
```

Notes

- Always test plug-in installations extensively to ensure that data collection is as expected before deploying in a production environment.
- Make sure you delete the cookie or use new, unique values during testing or variables will not be sent.

getVisitNum

The getVisitNum plug-in determines how many visits a user has made to your site and captures this number in a Analytics variable.

Plug-in Code and Implementation

CONFIG SECTION: No changes required for this section.

Plug-in Configuration

Place the following code within the `s_doPlugins()` function, which is located in the area of the `s_code.js` file labeled *Plugin Config*. Choose one Custom Traffic (`s.prop`) variable or one Custom Conversion (`s.eVar`) variable for use in capturing visit number data. This should be a variable that has been enabled using the Admin Console, but which is not currently in use for another purpose. You can use the following example and update it based on your requirements.

```
s.prop1=s.getVisitNum();
```



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should only be done by a developer with experience using and implementing Analytics.

PLUGINS SECTION: Add the following code to the area of the `s_code.js` file labeled PLUGINS SECTION. Do not make any changes to this portion of the plug-in code.

```
/*
 * Plugin: getVisitNum - version 3.0
 */
s.getVisitNum=new Function("tp","c","c2","","
+\"var s=this,e=new Date,cval,cvisit,ct=e.getTime(),d;if(!tp){tp='m';}\"
+\"if(tp=='m' || tp=='w' || tp=='d'){eo=s.endof(tp),y=eo.getTime();e.setTi\"
+\"me(y);}else {d=tp*86400000;e.setTime(ct+d);}if(!c){c='s_vnum';}if(!\"
+\"c2){c2='s_invisit';}cval=s.c_r(c);if(cval){var i=cval.indexOf('&vn=\"
+\"'),str=cval.substring(i+4,cval.length),k;}cvisit=s.c_r(c2);if(cvisi\"
+\"t){if(str){e.setTime(ct+1800000);s.c_w(c2,'true',e);return str;}els\"
+\"e {return 'unknown visit number';}}else {if(str){str++;k=cval.substri\"
+\"ng(0,i);e.setTime(k);s.c_w(c,k+'&vn='+str,e);e.setTime(ct+1800000);\"
+\"s.c_w(c2,'true',e);return str;}else {s.c_w(c,e.getTime()+'&vn=1',e)\"
+\";e.setTime(ct+1800000);s.c_w(c2,'true',e);return 1;}}\"");
s.dimo=new Function("m","y","","
+\"var d=new Date(y,m+1,0);return d.getDate();\"");
s.endof=new Function("x","","
+\"var t=new Date;t.setHours(0);t.setMinutes(0);t.setSeconds(0);if(x=\"\"
+\" 'm') {d=s.dimo(t.getMonth(),t.getFullYear())-t.getDate()+1;}else if(\"
+\" x=='w') {d=7-t.getDay();}else {d=1;}t.setDate(t.getDate()+d);return \"
+\"t;\"");
```

Parameters

- `tp` = (string, optional) Tracking period. use "d" for day, "w" for week, "m" for month, or a number for a custom number of days.
 - If day, week, or month is selected then the visit number will reset at the end of the day/week/month.
 - If a custom number of days is selected then the visit number will reset after that number of days.
 - If no value is provided then "m" will be used.
- `c` = (string, optional) Specify the persistent cookie name. Default is 's_vnum'.
- `c2` = (string, optional) Specify the session cookie name. Default is 's_invisit'

Returns

- returns the visit number (1,2,3,etc) of the visit. This number is incremented only on the first page of each visit.
- returns "unknown visit number" if the plug-in is unable to identify the visit number (cookies are blocked).

Examples

```
s.prop1=s.getVisitNum(365); //custom length, 365 days. Default cookie names
s.prop1=s.getVisitNum('w'); //resets weekly
s.prop1=s.getVisitNum('m','s_vmonthnum','s_monthinvisit'); //resets montly, custom cookie names
s.prop1=s.getVisitNum('d'); //resets daily
```

Notes

- Always test plug-in installations extensively to ensure that data collection is as expected before deploying in a production environment.
- This plug-in relies on the ability to set cookies in the user's web browser. If the user does not accept cookies, all visits will appear to be first visits.

hitGovernor

The s.hitGovernor plugin tracks the total number of Analytics image requests sent during a predefined rolling time frame, and can perform additional logic as required if that total exceeds a certain threshold.

Although traffic from bots, spiders, specific user agents, or a specific list of IP addresses can be identified as bot traffic or otherwise excluded from reporting, there can be traffic that is captured in your report suites that should not otherwise be counted. For example, a high number of clicks or page views during an unreasonable amount of time (i.e. approximately one request per second) could potentially be duplicitous traffic.

Using this plugin allows for that traffic to automatically be blocked for the remainder of that visitor's lifetime, and that traffic can also be dynamically identified within reports.

How the Hit Governor Plugin Works

The plugin increments a cookie value each time an image request is sent to your tracking servers, and tracks this over a rolling time frame. The default time frame is one minute, although that time frame can be overridden. (See [Implementation](#), below.) If the total number of hits during that time frame exceeds the default hit threshold (60), a final custom link image request is sent to set the *exceptionFlag* context data variable. The default hit threshold can also be overridden.

If desired, from that point forward, traffic can be prevented from being collected for that specific visitor for a default period of sixty days. Blocking the traffic requires an additional line of code in your doPlugins function, as outlined below. The time frame can be adjusted as well. The logic allows time to either include that visitor's IP address, User Agent, or Experience Cloud Visitor ID in the proper permanent exception logic, or to reset the timeout period after the sixty days have elapsed. If this traffic is identified as fraudulent by the plugin after sixty days, the traffic is again flagged as an exception and is not collected for another sixty days.

Reporting

No default variables or events need to be set up. However, we strongly recommend that you set up processing rules logic to set variables and events accordingly. Those custom variables and events might include:

- Experience Cloud Visitor ID
- IP Address
- User Agent

- **Flagged Exception Event**

Creating segments for those variables would then allow you to create segments and virtual report suites to view the overall site impact of those ambiguous hits.

We recommend using the values captured in the reporting to update the bot rules, DB VISTA rules, or company IP exclusions.

Internal Traffic

The Internal Traffic plugin dynamically identifies visitors originating from an internal network.

Identifying internal and external traffic promotes greater accuracy in all types of reporting, by providing a mechanism that filters and segments data being collected. Implemented correctly, it would also eliminate the need for a VISTA rule or Processing Rule that are typical approaches to identifying such traffic.

How does the Internal Traffic plugin work?

The plugin attempts to load a file that would only be available within your internal network/intranet, i.e. a 1x1 transparent pixel. If it is loaded successfully, traffic for that visitor would be identified as internal. Anything else would be external traffic.

Considerations

- The only downside to this approach is that a 404 error is displayed in the browser console for external visitors on the first page of their visit. This will not impact the user experience.
- We strongly suggest that you obtain approval from your Network or InfoSec team before trying to load a pixel hosted internally.
- Although the plugin will not move traffic to another report suite or exclude it from reporting (as might be done with a VISTA rule), custom logic can be included with its implementation, so that this functionality could take place client-side.

Implementation

1. **Add your Intranet Pixel:** You can add any type of file on your intranet that the plugin would attempt to access. A 1x1 transparent pixel is recommended. It should be placed in a location on your Intranet that is widely accessible from within your internal network(s).
2. **Configure an eVar:** An eVar will need to be added within your destination report suite. It should have an expiration of "Visit" and allocation of "Original Value (First)".
3. **Define the internal URL:** Within the AppMeasurement configuration variables and before doPlugins is instantiated, define the internal URL variable (s.intURL) for the pixel or other file can be used for the traffic check. For example:

```
s.intURL = "https://www.yourdomainhere.com/trafficCheck.gif"
```

4. **Modify doPlugins and set the eVar:** The plugin can then be initialized by including this line of code within the doPlugins section of your AppMeasurement library code, using the eVar defined in step one:

```
s.eVarXX = s.intCheck();
```

The variable value will be set to "internal" or "external".

5. **Add the Plugin Source Code:** Include the [plugin code](#) below the doPlugins section of your AppMeasurement file.

Plugin source code

Add this code below the doPlugins section of your AppMeasurement library.

```
s.intCheck=new Function("","
+\"var s=this;if(document.cookie.indexOf('intChk')===-1){try{document.\"
+\"cookie='intChk=1';var x=new XMLHttpRequest(),y;x.open('GET',s.intUr\"
+\"l,false);x.send();if(x.status===200&x.statusText==='OK'){y='intern\"
+\"al';}}catch(e){y='external'}finally{return y}}\"");
```

Other notes

- Always test plug-in installations to ensure that data collection happens as expected before deploying them in a production environment.
- Your implementation might be using a different object name than the default Adobe Analytics "s" object. If so, please update the object name accordingly.
- If you employ a Tag Management System, please follow its steps to update doPlugins and the other custom plugins.

performanceTiming

This plug-in operates by using the Navigation Timing JavaScript API for accurately measuring performance on the web. This provides a native method to get accurate and detailed timing statistics for page load events & asset load times. Previously, measurements of this sort have either utilized the JavaScript Date object for timing metrics, or a rudimentary extrapolation of the Navigation Timing metrics. Both methodologies, even though they provide some trended data for page load times, are unreliable.

What This Plug-In Does

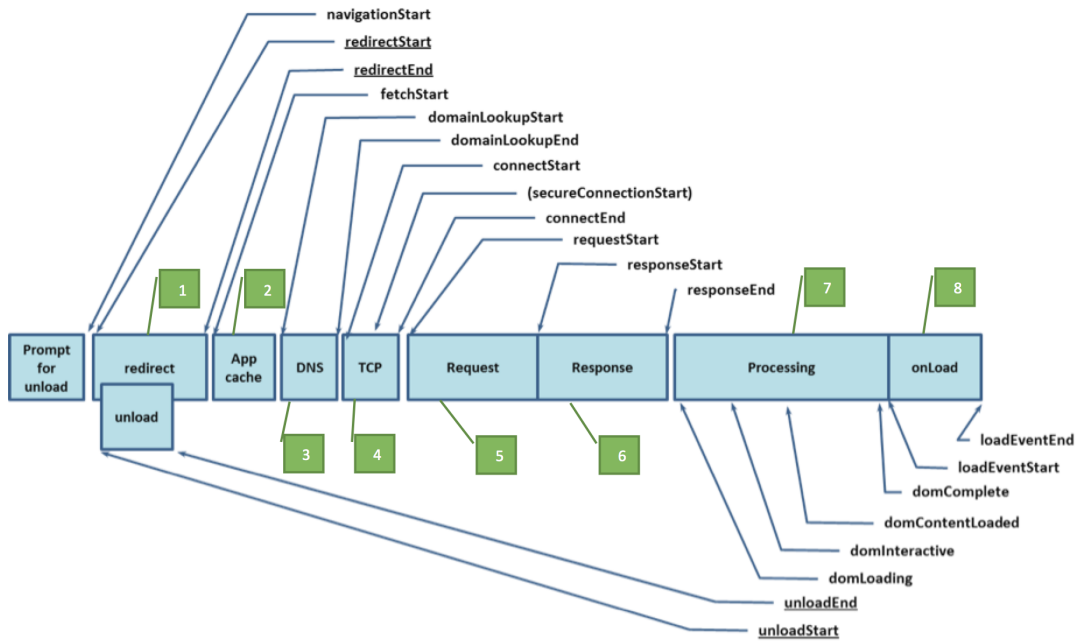


Important: This is a beta version of the plugin, and additional updates may be forthcoming.

This plug-in utilizes the following detailed events to track the individual timing components of a page load:

Event	Name	Calculated From
1	Redirect Timing	fetchStart - navigationStart
2	App Cache Timing	domainLookupStart - fetchStart
3	DNS Timing	domainLookupEnd - domainLookupStart
4	TCP Timing	connectEnd - connectStart
5	Request Timing	responseStart - connectEnd
6	Response Timing	responseEnd - responseStart
7	Processing Timing	loadEventStart - domLoading
8	onLoad Timing	loadEventEnd - loadEventStart
9	Total Page Load Time	loadEventEnd - navigationStart
10	Performance Instances	Counter

The following graph illustrates the timing attributes defined by the PerformanceTiming interface and the PerformanceNavigation interface with or without redirect, respectively.



Full details on the Navigation Timing object can be found here:

<http://www.w3.org/TR/navigation-timing/#sec-navigation-timing-interface>

In addition, the plugin can optionally use the performanceEntries object to record the asset name, asset load time start, and asset load time duration details for each individual asset loaded on a given page. A large amount of information is recorded with this plugin, and as such it requires that the DOM storage object is enabled in order to store the page load information between page views. Please be sure that your company's privacy policy allows for the use of the DOM storage object before enabling this functionality. It also requires the use of a listVar to track all assets.

Required Supporting Plug-Ins

- appendList
- getPreviousValue

Plug-In Code and Implementation



Note: The following instructions require you to alter the data collection code on your site. This can affect data collection on your site, and should be done only by a developer with experience using and implementing Adobe Analytics. This plugin is compatible only with AppMeasurement tracking libraries.

Config Section (before doPlugins):

`s.ptc`: Comma-separated list of events containing the 10 events you wish to use - the individual timing event components (events 1 - 8), the total page load time (event 9), and total performance instances (event 10) - in that specific order.

`s.ptc`: Set to determine whether or not to execute the plugin within doPlugins. Always set to false.

Sample Calls

```
s.ptc = 'event10,event11,event12,event13,event14,event15,event16,event17,event18,event19'
//[----- 1 to 8 -----][-- 9 --][- 10 -]
s.ptc = false;
```

doPlugins Section:

To initialize the plug-in, one line of code is required in the `doPlugins` section of your `s_code`, preferably after you have designated the `s.pageName` variable. If you wish to utilize the asset load time functionality within the plug-in, you must pass in the name of the list variable to be used. Otherwise, only the performance timing entries will be tracked in the events you previously specified in the `s.ptc` variable.



Note: In order to correlate performance timing entries with pages on your site, you must also initialize the `getPreviousValue` plug-in. We recommend comparing these performance entries with either the previous page name or the previous page URL value.

Sample Calls

```
/* Performance Timing */
s.eVar9 = s.getPreviousValue(s.pageName,'gpv_v9',''); //Record the previous page name in the
designated eVar of your choice
s.performanceTiming('list2')
```

Plugins Section:

Lastly, add the plug-in itself to your JavaScript implementation.

```
/* Plugin: Performance Timing Tracking - 0.11 BETA */
s.performanceTiming=new Function("v",""
+"var s=this;if(v)s.ptv=v;if(typeof performance!='undefined'){if(perf"
+"ormance.timing.loadEventEnd==0){s.pi=setInterval(function(){s.perfo"
+"rmanceWrite()},250);}if(!s.ptc||s.linkType=='e'){s.performanceRead("
+"");}else{s.rfe();s[s.ptv]='';}}");
s.performanceWrite=new Function("",""
+"var s=this;if(performance.timing.loadEventEnd>0)clearInterval(s.pi)"
+";try{if(s.c_r('s_ptc')== '&&performance.timing.loadEventEnd>0'){try{"
+"var pt=performance.timing;var pta='';pta=s.performanceCheck(pt.fetc"
+"hStart,pt.navigationStart);pta+='^^'+s.performanceCheck(pt.domainLo"
+"okupStart,pt.fetchStart);pta+='^^'+s.performanceCheck(pt.domainLook"
+"upEnd,pt.domainLookupStart);pta+='^^'+s.performanceCheck(pt.connect"
+"End,pt.connectStart);pta+='^^'+s.performanceCheck(pt.responseStart,"
+"pt.connectEnd);pta+='^^'+s.performanceCheck(pt.responseEnd,pt.respo"
+"nseStart);pta+='^^'+s.performanceCheck(pt.loadEventStart,pt.domLoad"
+"ing);pta+='^^'+s.performanceCheck(pt.loadEventEnd,pt.loadEventStart"
+"");pta+='^^'+s.performanceCheck(pt.loadEventEnd,pt.navigationStart);"
+"s.c_w('s_ptc',pta);if(sessionStorage&&navigator.cookieEnabled&&s.pt"
+"v!='undefined'){var pe=performance.getEntries();var tempPe='';for(v"
+"ar i=0;i<pe.length;i++){tempPe+='!';tempPe+=pe[i].name.indexOf('?)"
+">-1?pe[i].name.split('?')[0]:pe[i].name;tempPe+='|'+(Math.round(pe["
+"i].startTime)/1000).toFixed(1)+'|'+(Math.round(pe[i].duration)/1000"
+"").toFixed(1)+'|'+pe[i].initiatorType;}sessionStorage.setItem('s_pec"
+"',tempPe);}catch(err){return;}}catch(err){return;}}";
s.performanceCheck=new Function("a","b",""
+"if(a>=0&&b>=0){if((a-b)<60000&&((a-b)>=0)){return((a-b)/1000).toFix"
+"ed(2);}else{return 600;}}");
s.performanceRead=new Function("",""
+"var s=this;if(performance.timing.loadEventEnd>0)clearInterval(s.pi)"
+";var cv=s.c_r('s_ptc');if(s.ptc){var ela=s.ptc.split(',');}if(cv!='"
+"'){var cva=s.split(cv,'^^');if(cva[1]!=''){for(var x=0;x<(ela.lengt"
+"h-1);x++){s.events=s.apl(s.events,ela[x]+' '+cva[x],',',2);}s.event"
+"ts=s.apl(s.events,ela[ela.length-1],',',2);}s.linkTrackEvents=s.apl"
+"(s.linkTrackEvents,s.ptc,',',2);s.c_w('s_ptc','',0);if(sessionStora"
+"ge&&navigator.cookieEnabled&&s.ptv!='undefined'){s[s.ptv]=sessionSt"
+"orage.getItem('s_pec');sessionStorage.setItem('s_pec','',0);}else{s"
+"[s.ptv]='sessionStorage Unavailable';s.ptc=true;}}";
/* Remove from Events 0.1 - Performance Specific,
```

```

removes all performance events from s.events once being tracked. */
s.rfe=new Function("", "")
+"var s=this;var ea=s.split(s.events,',');var pta=s.split(s.pte,',');"
+"try{for(x in pta){s.events=s.rfl(s.events,pta[x]);s.contextData['ev"
+"ents']=s.events;}}catch(e){return;}";
/* Plugin Utility - RFL (remove from list) 1.0*/
s.rfl=new Function("l","v","d1","d2","ku",""
+"var s=this,R=new Array(),C='',d1=!d1?',':d1,d2=!d2?',':d2,ku=!ku?0:"
+"l;if(!l)return';L=l.split(d1);for(i=0;i<L.length;i++){if(L[i].inde"
+"xOf(':')>-1){C=L[i].split(':');C[1]=C[0]+''+C[1];L[i]=C[0];}if(L[i]"
+""].indexOf('=')>-1){C=L[i].split('=');C[1]=C[0]+''+C[1];L[i]=C[0];}"
+"if(L[i]!=v&&C)R.push(C[1]);else if(L[i]!=v)R.push(L[i]);else if(L[i]"
+""]=v&&ku){ku=0;if(C)R.push(C[1]);else R.push(L[i]);}C='';}return s."
+"join(R,{delim:d2})");

```

Notes

- Always test plug-in installations to ensure that data collection is as expected before deploying in a production environment.
- Because the plug-in passes the performance data as they are associated with the previous page, data is not collected for the final page view of the visit.
- If you are tracking asset timing, this plug-in relies on the ability to set DOM storage values in the user's web browser. If the user does not accept cookies and have DOM storage enabled, the plug-in will not pass data into Analytics.
- A very small percentage of users will not pass navigation timing data due to browser limitations, and logic is contained within the plugin to ensure that the data is not skewed as a result – particularly with a small portion of mobile browsers. However, this plug-in has been successfully tested in IE, Firefox, Chrome, and Safari.
- Calculated metrics should be created to aid in summarizing and understanding visitor behavior associated with these metrics:
 - Average Redirect Timing (Redirect Timing/Performance Timing Instances)
 - Average App Cache Timing (App Cache Timing/Performance Timing Instances)
 - Average DNS Timing (DNS Timing/Performance Timing Instances)
 - Average TCP Timing (TCP Timing/Performance Timing Instances)
 - Average Request Timing (Request Timing/Performance Timing Instances)
 - Average Response Timing (Response Timing/Performance Timing Instances)
 - Average Processing Timing (Processing Timing/Performance Timing Instances)
 - Average onLoad Timing (onLoad Timing/Performance Timing Instances)
 - Average Page Load Timing (Total Page Load Time/Performance Timing Instances)

trackTNT

Collects the information for the Target to Analytics integration. This plug-in is replaced by the Adobe Analytics and Adobe target integration.

See [Adobe Analytics and Adobe Target integration](#).

Pathing

Pathing is defined as the path that users take through your site.

For example, a visitor went to page A, then page B, then page C. Pathing is one of the very powerful features of Analytics. The tremendous insight it brings is critical to businesses looking to understand visitor traffic patterns.

Out-of-the-box Analytics provides pathing at the page level. The basic idea behind pathing is you are shown the order of pages that users saw during their visits. This data is presented in several different reports that format the data in different ways, depending on what you are trying to see.

 **Note:** To enable pathing, go to **Admin > Report Suites > Edit Settings > Traffic > Traffic Variables**. To enable pathing on the Site Section and Server reports, contact Customer Care.

Enabling Pathing on a prop

Though pathing reports are available out-of-the-box for pages, pathing can also be enabled for **custom traffic** variables.

Enabling pathing on a prop can be done in the Admin Console. The following list contains few examples of a business question that can be solved by enabling pathing for a prop.

- Where does a user go on my site after they come to my site from a campaign?
- What are the common site sections my users go to in their visits?
- How do I see page pathing by user type?

Advanced pathing is available in Ad hoc analysis. Ad hoc analysis provides segmentation on any report on the fly without needing to enable this setting, or structure data in the prop in certain ways. It is the easiest and most optimal way to segment your path reports as well. For more information see the [Ad hoc analysis documentation](#).

When pathing is enabled for a prop, you get a new set of pathing reports that are found below the standard pathing reports. Pathing reports tell you the order that it saw the pages come in for that visitor. When you enable pathing on a prop, it does the same thing: it tells you the order of the prop values it saw come in for the user. This order of values is displayed in different ways depending on the path report you are viewing.

 **Note:** To enable pathing, go to **Admin > Report Suites > Edit Settings > Traffic > Traffic Variables**. To enable pathing on the Site Section and Server reports, contact Customer Care.

Pathing by Campaign or Tracking Code

Helps you answer the question, "After a user clicks into my site from a campaign, where do they go on my site?"

To answer this question, you need to set aside an **sprop** to be used for this report. You then need to structure the data to populate into the prop in such a way that it makes sense when pathing is enabled.

For this example, you are going to use **prop1** as your campaign pathing prop. Now you want to populate this **sprop** with the same value you put in the **page name** variable. See below:

```
s.prop1=s.pageName;
```

You should do this on all pages unless the person has clicked from a campaign. If they have clicked from a campaign and are on the landing page of the campaign, then you populate the prop with a concatenation of the campaign and the **pagename**. See below:

```
s.prop1=s.campaign + ' : ' + s.pageName;
```

If the campaign they clicked was named "banner1234," and the page it landed on was named "Home Page," the value in that prop would be "banner1234 : Home Page." On every subsequent page you put the **pagename** in the prop as shown above.

When a user clicks this campaign and views four total pages in that visit, you get the following values in the sprop in this order:

```
"banner1234 : Home Page" > "Page 2" > "Page 3" > "Page 4"
```

With our data captured in **prop1** in this way, with pathing enabled on this prop, you can now look at one of various pathing reports to understand how they path through the site after they click from a campaign.

You can specify the start page from which to start the path report. In this case, you selected "banner1234 : Home Page", because you want to know where users went after they clicked from campaign "banner 1234." This report is only an example of one of many path reports.

Reasons Pathing may not be Recorded

List of reasons pathing information might not be recorded and displayed in reporting.

- Pathing has not been enabled for that prop in that report suite. This enablement is report suite specific.
- Data is not being passed into the correct prop.
- Pathing has been enabled and the data is in the prop, but it hasn't shown up in the reports. Though the **sprop** data may show up within 10 minutes, the pathing data will not show up until the visitor's session has ended. It ends after 30 minutes of inactivity. Analytics then takes another 10 minutes to finish the processing of the path data for final presentation in reports.

If you have checked all of these and the data is still not showing up, check with Customer Care for further debugging.

Moving from Section to Section

If you want to know how visitors move from section to section on your site, you first need to ensure that you are tagging your sections with the *channel* variable.

With this data in place, you must have pathing enabled on the *channel* variable. If a user starts on the home page and moves to the Sports section, then to the News section, this activity shows up in the **Site Section** path reports. These reports display after pathing is enabled on this variable.

Moving from Page Template to Page Template

If your site has types of pages or page templates, you could use pathing to understand how they move from type to type.

You need to first capture the page type or template in a prop set aside for only that purpose. Then have Adobe Customer Care enable pathing on that prop. You can now view your page template pathing reports to understand how users move from template to template on your site.

Segmenting Paths by User Type

Segmenting paths by user type is a common request for trying to understand how specific user types path on your site.

You can concatenate the user type and page name into a **sprop** and enable pathing on the **sprop**.

For example, let's say you have two user types: "Registered" users and "Non-Registered" users. You need to distinguish between these two user types on each page and put these values into your designated **sprop**. When you populate the prop, it should display as shown below:

```
s.prop1="Registered : " + s.pageName;
```

If the user is a registered user and visited the home page, the value in the prop displays as follows:

```
"Registered : Home Page"
```

If they click to another page named "Page 2", the value on that page displays as follows:

```
"Registered : Page 2"
```

With **Pathing** turned on, you see those two values in succession. If you want to know how registered users move from the home page, find the value "Registered : Home Page" in one of the path reports and see the next pages they visited. In this case, they next went to "Page 2."

Purchase Events

For the **purchase** event, Analytics variables are used to capture specific purchase information. The **s.purchaseID** variable is used to serialize (de-duplicate) the event.

The *purchaseID* is limited to 20 characters. The *purchaseID* variable serializes all events passed in the variable **s.events**, and overrides any serialization value for events. If *purchaseID* is left blank, each instance of the purchase event, even page reloads, is counted.

If a purchase event is called without a *purchaseID*, a unique one is automatically generated based on the *s.products* and *s.events* variables. This automatically generated *purchaseID* is stored locally as a cookie value within the visitor's browser, and is not sent to any report suite tables. Manually defined *purchaseIDs* on the other hand are sent to a back-end table within the report suite. The last five purchases made by a visitor (with or without *purchaseID*) are stored in the local cookie.

When a visitor makes any purchase, the following checks are made:

- Does the *purchaseID* match any of the five cookie values? If so, the image request is considered a duplicate purchase. All conversion variables, including the purchase event, do not appear in reporting.
- If *purchaseID* is defined, does it match any value in the report suite's back-end table? If so, the image request is considered a duplicate purchase. All conversion variables, including the purchase event, do not appear in reporting.
- If the *purchaseID* is unique to both the last 5 stored values in the cookie and the report suite's *purchaseID* table, the image request is unique and included in reporting. For example, if five purchases are already included in the local cookie, the oldest one is overwritten.

Specific server-side code can be used to generate the unique number (alphanumeric value) embedded in the HTML source. Usually the Order ID, or similar alphanumeric value, is used for this purpose. This value should not change if the user refreshes the page. The following is a short example (note the *purchaseID* code in bold):

Syntax

```
s.products="Category;ProductName;Qty;total_price [,Category2;ProductName2;Qty;total_price]"
s.state="XX"
s.zip="00000"
s.purchaseID="<%=getPurchaseID( )%>"
s.events="purchase"
```

Examples

```
s.products="Footwear;Hiking Boots (1234);1;170.00"
s.state="UT"
s.zip="84097"
s.purchaseID="12341234"
s.events="purchase"
```

Additional Notes

- Be sure to use server-side code to generate the unique identifier for the event. Do not use a JavaScript randomization function to generate a number, which generates unique numbers each time the page is loaded (each **instance/pageview** counts).
- The unique identifiers are applicable to all users across all sessions. Therefore, ensure the identifier is unique across users and sessions. For instance, if the Order ID repeats after 30 days, append the date of the order to make the **Order ID** sufficiently unique.

- The serialization value may be alphanumeric values up to 20 characters in length. This is identical to the limitations of **s.purchaseID** (replace s. with s_ for G Code).
- The **s.purchaseID** variable serializes all events passed in the variable **s.events**, and overrides any serialization value for events. Do not use **Event serialization** for any events if the **s.purchaseID** variable is used on the current page (replace s. with s_ for G Code).

Event Serialization

Event serialization is the process of implementing measures to prevent duplicate events from entering Analytics reporting. This can commonly occur when a user refreshes the page multiple times, navigates to a certain page multiple times, or saves the web page to their computer (for example, if a customer saves a purchase confirmation page to their computer, every time they view it orders and revenue would be counted again if event serialization was not in place).

Event serialization is useful in the following instances:

- A page may be reloaded or refreshed and repeatedly send an event. **Event serialization** prevents events from being recounted by using a serial number for each event.
- The user saves the page to his/her machine for later review. This scenario is quite common on purchase confirmation pages to review purchase receipts. **Event serialization** prevents the subsequent page reloads from re-counting the events.



Note: *Data Sources does not support event serialization or de-duplication.*

This document describes the process used to implement **Event serialization** for **conversion** and **custom** events. To use **Event serialization**, you must first enable it in **Admin > Report Suite > [select report suite] > Edit Settings > Success Events**. Then select which events you want recorded in the **Unique Event Recording** column.

Default Behavior

The default behavior is to count each instance of an event. An event is set for each **pageview** that is counted, even on page reloads or page refreshes. The **s.purchaseID** variable is used in order to uniquely identify each order (purchase). This allows a user to reload the order page without recounting the order, revenue, or products. A similar feature is available for all events. This includes pre-defined events such as **prodView** and **scCheckout**, as well as all custom events.

Methods of Event Serialization

There are two methods of keeping events from counting more than once.

To use **Event serialization**, you must first enable it in **Admin > Report Suite > [select report suite] > Edit Settings > Success Events**. Then select which events you want recorded in the **Unique Event Recording** column. There are three different settings an event can be set to.

Always record event: This is the default behavior of all events when initially enabled. All events included in image requests will be sent directly to Analytics, including page reloads.

Record once per visit: An event with this setting enabled will only track the first instance of that event in a given visit. Once a new visit starts, each event with this setting enabled can be tracked again. This is an effective way to mitigate duplicate events via browser refreshes.

Use event ID: This setting allows the capability to associate each event with a unique ID. If Analytics sees an eventID it has already seen before with that variable, it will not be counted in reporting.

The only event that cannot have event serialization enabled is the purchase event, as this uses the `s.purchaseID` variable. All other eCommerce events and custom events can have these settings enabled.

- Use a unique identifier to let an event fire once per unique ID.
- Send multiple events, then use configure Analytics to let an event fire once per visit.

To implement **Event serialization**, provide a unique ID for the event, for example `event1:1234ABCD`.

Event Serialization - Once per Unique ID

Once **Event serialization** is implemented, and Analytics receives a duplicate number, it ignores the event. An event is counted only once per unique value. If the number is unique, another event instance is counted, as shown in the following example:

User Name	Description	Event Syntax	Analytics Total Event1 Count
John	User views page for the first time.	<code>event1:1000</code>	1
John	User reloads the page (a form submit may fail, and cause the page to reload).	<code>event1:1000</code>	1
Stacy	User views page for the first time.	<code>event1:1001</code>	2
Stacy	User reloads the page (a form submit may fail, and cause the page to reload).	<code>event1:1001</code>	2
Jill	User views page for the first time, enters information correctly, and moves on to next page.	<code>event1:1002</code>	3
Jamie	User views page for the first time	<code>event1</code>	4
Jamie	User forgets to fill in the last name field on the form. The form is displayed again with the missing information highlighted.	<code>event1</code>	5

s.events Syntax

Use the following syntax for serializing events. In the first example, `event1` is serialized, while `event2` is not serialized (an instance is counted for each **pageview** or page reload).

```
s.events="[event]:[serial number]"
```

Samples:

```
s.events="event1:12341234"
s.events="event1:12341234,event2"
s.events="purchase,event1:12341234"
```

Serializing a counter event with a value other than 1

Add the serial number after the event value similar to the following:

```
s.events="event1=100:12341234"
```

Note the following when selecting serialization IDs:

- Serialization IDs must be 20 characters or less.
- Serialization IDs must be alphanumeric characters.
- Serialization IDs are separate for each success event. Therefore, you can use the same ID for multiple success events if needed, just not for the same success event.

- Serialization IDs are tied to the report suite, so if you are using multi-suite tagging to send data to multiple report suites, keep this in mind.
- Serialization IDs never expire, so even if the same ID is used years later, the success event will not be counted again.
- Cookie deletion does not prevent Event ID serialization because serialization IDs are stored on Adobe servers and are not cookie-based.
- The one success event with which it is not possible to use Event ID serialization is the Purchase event, which uses a special `s.purchaseID` variable for serialization.

Event Serialization - Once per Visit

Analytics offers a feature to let an event only fire once per visit.

This can be enabled from the UI: **Admin > Report Suite > Edit Settings > Conversion > Success Events.**

Serializing eVars

The same functionality used to keep an event from being fired more than once per visit can be applied to **event** instances. When enabled for an **event**, **click-throughs** and **instances** are only counted once per visit for a specific **event**. This is not counting once per value, but once per **event**. This means that if an **event** or *campaign* variable is set to record once per visit, only the first value seen in a visit shows a **click-through** or **instance**. However, if the **event** is set to allocate credit to the most recent value, then the most recent value still receives credit, even if it does not have an **instance**. The following example helps to illustrate this point.

If **s.campaign** is set to **Record once per visit**, and within a single visit 20, pages are viewed. First, there are 10 pages where **s.campaign** is set to "abc," then 10 pages are viewed where **s.campaign** is set to "xyz." On all 20 pages, **event1** is fired. The following screen shot illustrates the results. Notice that there are no **instances** or **Click-throughs** associated with "xyz," but it does receive credit for all events fired.

Details

Tracking Code	Click-throughs	Custom 1
1. abc	1 100.0%	10 50.0%
2. xyz	0 0.0%	10 50.0%
Total	1	20

Identifying Unique Visitors

Adobe uses a cookie to track unique browsers/devices.

Analytics Visitor ID Order

Adobe Analytics provides several mechanisms to identify visitors. The following table lists the different ways a visitor can be identified in Analytics (in order of preference):

Order Used	Query Parameter (collection method)	Present When
1	<i>vid (s.visitorID)</i>	s.visitorID is set
2	<i>aid (s_vi cookie)</i>	Visitor had an existing s_vi cookie before you deployed the Visitor ID service, or you have a visitor ID grace period configured.

Order Used	Query Parameter (collection method)	Present When
3	<i>mid (AMCV_ cookie set by Experience Cloud Visitor ID service)</i>	Visitor's browser accepts cookies (first-party)
4	<i>fid (fallback cookie on H.25.3 or newer, or AppMeasurement for JavaScript)</i>	Visitor's browser accepts cookies (first-party)
5	<i>IP Address, User Agent, Gateway IP Address</i>	Visitor's browser does not accept cookies.

In many scenarios you might see 2 or 3 different IDs on a call, but Analytics will use the first ID present from the previous table as the official visitor ID. For example, if you are setting a custom visitor ID (included in the "vid" query parameter), that ID will be used before other IDs that might be present on that same hit.



Note: Each Analytics visitor ID is associated with a visitor profile on Adobe servers. Visitor profiles are deleted after at least 13 months of inactivity regardless of any visitor ID cookie expiration.

Custom Visitor ID

You can implement a custom method to identify visitors by setting the `s.visitorID` variable.

A custom Visitor ID can be used on sites where you have unique way to identify visitors. An example of this is an ID generated when a user logs in to web site using a user name and password.

Should you have the ability to derive and manage the **visitor IDs** of your users, you can use the following methods to set the ID:

Method	Description
<i>s.visitorID</i> variable	If JavaScript is used on the browser, or if you are using any other AppMeasurement library, you can set the visitor ID in a data collection variable.
Query string parameter on the image request	This lets you pass the visitor ID to Adobe via the vid query string parameter on a hard-coded image request.
Data Insertion API	On devices using wireless protocols that don't accept JavaScript, you can send an XML post containing the <code><visitorid/></code> XML element to Adobe collection servers from your servers.
URL re-writing and VISTA	Some deployment architectures provide support for using URL re-writing to maintain session state when a cookie cannot be set. In such cases, Adobe engineering services can implement a VISTA rule that would look for the session value in the URL of the page, then format and place into the visid values.

: Custom Visitor IDs must be sufficiently granular/unique.

An invalid implementation of custom Visitor IDs can lead to incorrect data and poor reporting performance. If the custom Visitor ID is not unique or granular enough, or is incorrectly set to a common default value such as the string "NULL", or "0", the hits from many different visitors will be seen by Adobe Analytics as a single visitor. This situation results in incorrect data, with visitor counts being too low and segments not working correctly for that visitor. An insufficiently granular custom Visitor ID also prevents the data from being properly spread across nodes in the Analytics reporting cluster. In this situation, one node becomes overloaded and cannot process report requests in a timely fashion. Eventually all reporting for the report suite will fail. Poorly implemented custom Visitor IDs might not immediately impact reporting performance because Analytics can often handle several months' worth of unbalanced

data; however, over time a poorly implemented custom Visitor ID value can become problematic to the point of requiring Analytics to disable processing for affected report suites.

Implementers should follow the guideline that a single custom Visitor ID value should never be credited for more than 1% of your report suite's traffic. Although the 1% guideline is sufficient for most report suites, the actual limit that might cause reporting performance to be impacted may be lower than 1% for very large report suites.

Experience Cloud ID Service

The Experience Cloud ID service replaces the legacy Analytics visitor ID mechanism, and is required by heartbeat video measurement, Analytics for Target, and future Experience Cloud core services and integrations.

See [Experience Cloud ID Service](#) for product documentation about this service.

Analytics Visitor ID

When a user visits your site, a persistent cookie is set by the Adobe web server by including it in the HTTP response to the browser. This cookie is set on the specified data collection domain.

When a request is sent to the Adobe data collection server, the header is checked for the visitor ID cookie (`s_vi`). If this cookie is in the request, it is used to identify the visitor. If the cookie is not in the request, the server generates a unique visitor ID, sets it as a cookie in the HTTP response header, and sends it back with the request. The cookie is stored in the browser and is sent back to the data collection server during subsequent visits the site, which enables the visitor to be identified across visits.

Third-Party Cookies and CNAME records

Some browsers, such as Apple Safari, no longer store cookies that are set in the HTTP header from domains that do not match the domain of the current website (this is a cookie used in a third-party context, or a third-party cookie). For example, if you are on `mysite.com` and your data collection server is `mysite.omtrdc.net`, the cookie that is returned in the HTTP header from `mysite.omtrdc.net` might be rejected by the browser.

To avoid this, many customers have implemented CNAME records for their data collection servers as part of a [first-party cookie implementation](#). If a CNAME record is configured to map a hostname on the customer's domain to the data collection server (for example, mapping `metrics.mysite.com` to `mysite.omtrdc.net`), the visitor ID cookie is stored since the data collection domain now matches the domain of the website. This increases the likelihood that the visitor ID cookie will be stored, but it introduces some overhead as you need to configure CNAME records and maintain SSL certificates for data collection servers.

Cookies on Mobile Devices

When mobile devices are tracked using cookies, there are some settings you can use to modify how measurement occurs. Cookie default lifetime is 5 years, but you can use the CL query param variable (`s.cookieLifetime`) to change this default. To set the cookie location for cname implementations, use the CDP query string `s.cookieDomainPeriods`. The default is 2 if no value is specified, and the default location is `domain.com`. For implementations that do not use CNAME, the visitor ID cookie location is at the `207.net` domain.

Fallback ID Methods

If other visitor ID methods fail, Adobe sets a fallback cookie or uses a combination of IP address and user agent to identify the visitor.

Fallback Visitor Identification Method

AppMeasurement for JavaScript 1.x and JavaScript H.25.3 (released January 2013) contain a new fallback visitor identification method for visitors whose browser blocks the cookie set by Adobe's data collection servers (called `s_vi`). Previously, if a cookie could not be set, visitors were identified using a combination of the IP address and user agent string during data collection.

With this update, if the standard `s_vi` cookie is unavailable, a fallback cookie is created on the domain of the website with a randomly generated unique ID. This cookie, named `s_fid`, is set with a 2 year expiration and is used as the fallback identification method going forward. This change should result in increased accuracy in visit and visitor counts in situations where the primary cookie (`AMCV_` or `s_vi`) cannot be set.

Visits total includes all visitors that are identified by the `s_vi` cookie or by using a fallback method.

IP Address, User Agent, Gateway IP Address

. If the `AMCV_` or `s_vi` and the `s_fid` cookies cannot be set, visitors are identified using a combination of IP address and User Agent.

Identifying Mobile Devices

Most mobile devices accept browser cookies. However, in cases when devices do not accept cookies, another method is used to uniquely identify wireless devices.

Adobe has identified a number of HTTP [subscriber ID headers](#) that uniquely identify a majority of mobile devices. Those headers often include the device phone number (or a hashed version of the number), or other identifiers. The majority of current devices have one or more of the headers that uniquely identify the device, and all Adobe data collection servers automatically use those headers in place of a Visitor ID.

In a typical image request, a '1' in the path (`/b/ss/rsid/1`) causes Adobe servers to return a gif image and to attempt to set a persistent **visitor ID** cookie (`AMCV_` or `s_vi`). However, if the device is recognized as a mobile device based on the HTTP headers, a '5' is passed in place of the '1', which indicates that a wbmp format image should be returned and that our list of recognized wireless headers (not a cookie) should be used to identify the device.

The following table lists out the order of the ID methods used based on the return image type value ('1' or '5') in the path:

Setting	ID method order
/1/	Default: <ul style="list-style-type: none"> • Custom visitor ID • Cookie • Subscriber ID Header • IP Address-UserAgent-Gateway IP Address
/5/ /5.1/ /5.5/	Device was identified as a wireless device, or /5/ was manually sent in the image request: <ul style="list-style-type: none"> • Custom visitor ID • Subscriber ID Header • Cookie • IP Address-User Agent-Gateway IP Address

You can also pass a '1' or a '5' in manual image requests, but be aware that these codes are mutually-exclusive, therefore always passing '5' does not leverage a cookie when it is supported. You can incorporate your own mechanism to determine if a device supports cookies, and if so, pass a '1' in the image rather than a '5'. The accuracy improvement in this situation is limited to the number of mobile devices supporting cookies.

Subscriber ID Headers

The subscriber ID method is generally more reliable than a cookie for user identification because of cookie deletion, cookie acceptance issues, and gateway cookie management issues.

You can improve changes of identifying a visitor by being added to the white list for the carrier that your mobile visitors use. To get access to the carrier's visitor ID, contact the carrier to add your domain to their white list. If you are on a carrier's white list, you also have access to subscriber ID headers that you may not otherwise be able to access.

The following list of headers is used to identify wireless devices. The algorithm for processing the headers is to

1. extract the HTTP header key (the name of the header, such as,"X-Up-Calling-Line-ID")
2. trim out all non alpha (A-Z and a-z) characters
3. convert the header key to lowercase
4. compare the end of the key to the ones in the following table to find a match:

Header	Type	Example
callinglineid	ID	X-Up-Calling-Line-ID: 8613802423312
subno	ID	x-up-subno: swm_10448371100_vmag.mycingular.net
clientid	ID	ClientID: eGtUpsqEO19zVHmbOkgaPVI-@sprintpcs.com
uid	ID	x-jphone-uid: a2V4Uh21XQH9ECNN
clid	ID	X-Hts_clid: 595961714786
deviceid	ID	rim-device-id: 200522ae
forwardedfor	ID or IP Address	X-Forwarded-For: 127.0.0.1
msisdn	ID or IP Address	X-Wap-msisdn: 8032618185
clientip	IP Address	Client-ip: 10.9.41.2
wapipaddr	IP Address	X-WAPIPADDR: 10.48.213.162
huaweinasip	IP Address	x-huawei-NASIP: 211.139.172.70
userip	IP Address	UserIP: 70.214.81.241
ipaddress	IP Address	X-Nokia-ipaddress: 212.97.227.125
subscriberinfo	IP Address	X-SUBSCRIBER-INFO: IP=10.103.132.128

For example "callinglineid" would match "X-Up-Calling-Line-ID" and "nokia-callinglineid." The header type tells us what to expect in the header. The order of header priority is listed here (if a "callinglineid" header is present it is used instead of "subno").

You can use [Dynamic Variables](#) to extract specific values from a header.

Cross-Device Visitor Identification

Cross-device visitor identification helps you connect visitors across multiple devices.

For information about identifying visitors across devices, refer to the [Adobe Experience Cloud Device Co-op Documentation](#).

Connecting Users Across Devices

Cross-device visitor identification uses the **visitor ID** variable, `s.visitorID`, to associate a user across devices.

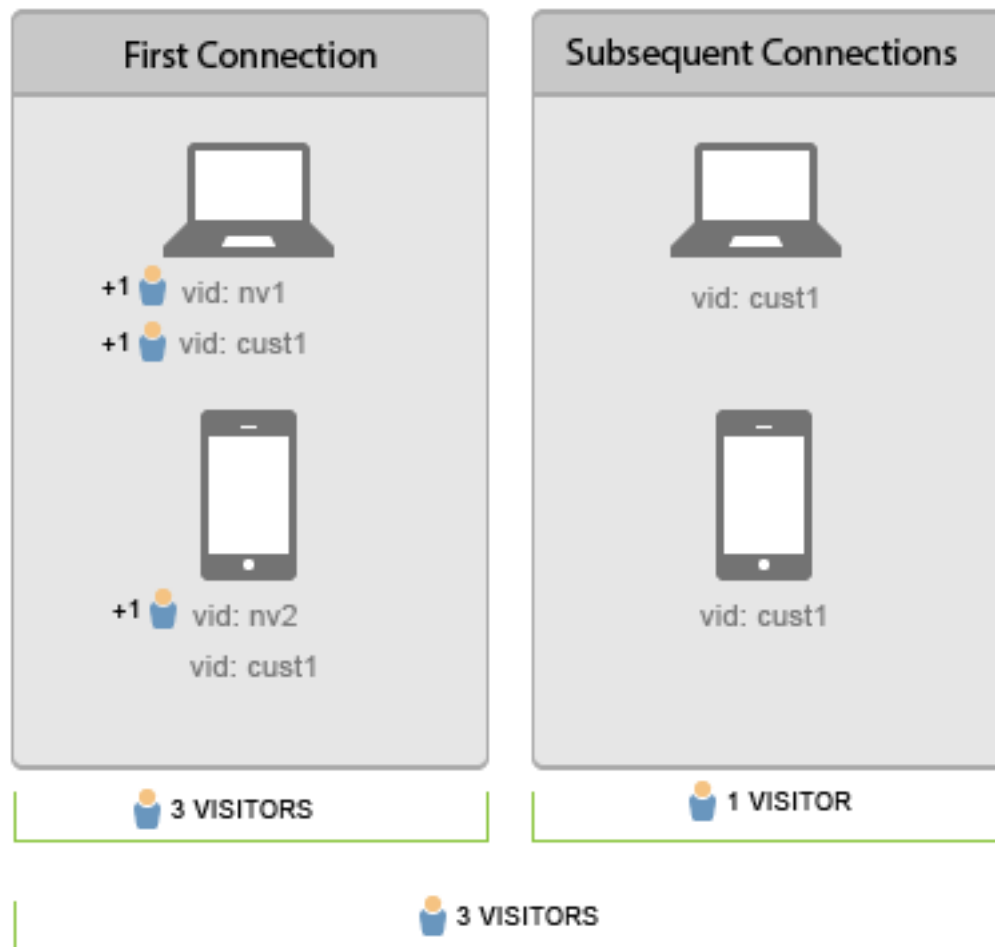
When you provide a **visitor ID** variable with a hit, the system checks for any other visitor profiles that have a matching **visitor ID**. If one exists, the visitor profile that is already in the system is used from that point forward and the previous visitor profile is no longer used.

The **visitor ID** is typically set after authentication, or after a visitor performs some other action that enables you to uniquely identify them independently of the device that is used. We recommend creating a hash of the username or an internal ID that does not contain any personally identifiable information.

In the [previous example](#), after the customer signs on from each device, they are all associated with the same user profile. If the visitor later signs out on a device, stitching continues to work since the **visitor IDs** that are stored in a cookie on each device are already associated with the same visitor profile. We recommend populating the `s.visitorID` variable whenever possible in case the **visitor ID** cookie is deleted.

Unique Visitor and Visits Counts

Consider the following connection sequence for two devices:



On the first data connection

- Visitor de-duplication is not retroactive.

After authentication on the laptop, hits with either visitor ID (`nv1` or `cust1`) will be considered the same individual by Adobe Analytics. However, visitor de-duplication is not retroactive, so 2 unique visitors are counted.

On the first data connection on the mobile device the customer is not recognized so a new unique visitor is counted. Once the user is authenticated (`cust1`) on the mobile device, Adobe Analytics maps `cust1` back to the visitor ID provided on the main site, so no more unique visits are incremented.

Each new device or browser authenticated will add 1 unique visitor.

On subsequent data connections

On subsequent data connections to authenticated devices unique visitors are not incremented.

Data Impact of Cross-Device Visitor Identification

Overview of how enabling cross-device visitor identification affects the data that you see in reports.

To understand how this feature affects data collection, it is useful to understand the visitor data fields in a visitor profile:

Data Field	Description
Visitor ID cookie	ID generated automatically on the first visit from a device or browser and stored in the <code>s_vi</code> cookie.
Visitor ID variable	Optional visitor ID that is set using the <code>s.visitorID</code> variable. This value is populated after a user authenticates and might match an ID that your company uses to track a user across multiple digital marketing channels.
Effective Visitor ID	The effective visitor ID is the actual ID for the user profile. This value is set to the visitor ID cookie, or to the visitor ID variable if one is provided.

Example visit

Example containing a sample of server calls sent in a common customer interaction.

Server Call	Action	Visitor ID Cookie	Visitor ID Variable	Effective Visitor ID	Visit Page Number	Visit Number
1	A visitor clicks a link in a marketing email and visits your site from home computer. This visitor has visited your site 7 other times in the past.	1	-	1	1	8
2-8	Visits 7 additional pages on your site.	1	-	1	2-8	8
9	Authenticates on home computer.	1	CID1	CID1	9* * This is CID1's first hit ever, so it takes over and continues on the visitor profile from Visitor ID 1.	8
10	Visits 1 additional page.	1	CID1	CID1	10	8
11	Opens site from laptop at office. This	2	-	2	1	1

Server Call	Action	Visitor ID Cookie	Visitor ID Variable	Effective Visitor ID	Visit Page Number	Visit Number
	visitor has not visited your site before using this device.					
12	Authenticates on laptop.	2	CID1	CID1	1	9
13	Views 1 additional page.	2	CID1	CID1	2	9

Visitors

Analytics counts each unique effective **visitor ID** as a unique visitor.

If you look at the [previous table](#), this occurred 3 times: at hits 1, 9, and 10. This occurs because the effective **visitor ID** is the same for both server calls, and occurs even though the visits might be several hours apart and on different devices.

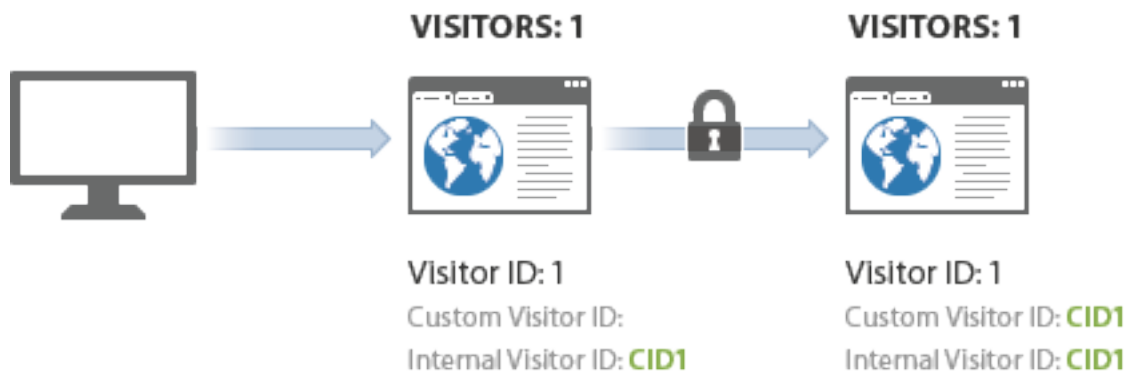
This can increase the number of unique visitors you see when cross-device visitor identification is enabled. The visitor might be counted twice on the same visit: once for the initial visit and again after the user is authenticated.

When a new visitor views your site, the `s_vi` cookie is populated and stored. On the data collection server, a new visitor profile is created for this visitor ID, and the effective **visitor ID** on the profile is set to match the cookie.

When cross-device visitor identification is enabled, if a **visitor ID** variable is provided in a subsequent hit (for example, after authentication), the effective **visitor ID** is updated to match the custom value. This can cause the effective **visitor ID** to change directly after authentication, resulting in multiple visitor counts.



After the initial association, visit counts return to normal because the visitor is associated through the **visitor ID** cookie. If the visitor later views your site and then authenticates, the visitor count is not inflated because the effective **visitor ID** doesn't change after authentication.



Visits

Analytics counts a visit each time a server call occurs with a **Visit Page Number** equal to 1.

If you look at the [previous table](#), this occurred 4 times: at hits 1, 9, 11, and 12. Similar to visitors, this value returns to normal after the initial association because the **Visit Page Number** is reset back to 1 due to a change in the effective **visitor ID**.

Create Segments

You can create a segment whenever an association occurs for a given **visitor ID** cookie.

Based on the [previous table](#), if you created a segment where visit number equals 9, it would include server calls 12 and 13. Even though server call 11 was technically part of the same visit, the historical data for that server call is not altered and the visit number remains unchanged.

Geo-Segmentation Data

Geo-Segmentation data is recorded based on the first hit of the visit, and does not change for a single visit regardless of the device used.

If a customer browses your site from his or her home computer, and then from a mobile device within 30 minutes, the Geo-Segmentation data is not changed. If you are using VISTA to populate an eVar with Geo-Segmentation data, it is based on the IP address in each hit. This could result in multiple Geo-Segmentation data values if the IP address changes for the same visit.

Attribution and Persistence

When visitor profiles are merged after being associated with the same **visitor ID** variable, attribution is not changed in the historical data set.

- When the variable `s.visitorID` is set and sent on a hit, the system checks for any other visitor profiles that have a matching visitor ID.
- If a profile exists, the visitor profile that is already in the system is used from that point forward and the previous visitor profile is no longer used.
- If no matching visitor ID is found, a new profile is created.

When an un-authenticated customer first arrives at your site, that customer is assigned a visitor profile by Adobe Analytics. As shown in [Unique Visitor and Visits Counts](#), on authentication a new profile is created. When the new profile is created, one visit ends and another visit starts.

On the first data connection

The example below is a representation of how data is sent to Adobe Analytics when a customer authenticates for the first time, on the first device:

- eVar16 has an expiration of 1 day and evar17 expires on visit.
- The `post_visitor_id` column represents the profile maintained by Adobe Analytics.
- The `post_evar16` and `post_evar17` columns show shows the persistence of eVars.
- `cust_visid` represents a value set in `s.visitorID`.
- Each row is one 'hit', a single request sent to Adobe Analytics data collection servers.

date_time	visitor id	cust_visid	post_visitor_id	visit_num	vis
2/06/2012 15:33	sv1		258887744665544	1	
2/06/2012 15:38	sv1		258887744665544	1	
2/06/2012 15:39	sv1	u999	2947539300	1	
2/06/2012 15:42	sv1		2947539300	1	
2/06/2012 15:42	sv1		2947539300	1	

On the first data connection containing a previously unrecognized `s.visitorID` value (u999 above), the new profile is created. Persistent values from the previous profile are transferred to the new profile.

- eVars set to expire on the visit are not copied to the authenticated profile. Note the value `car` above is not persisted.
- eVars set to expire by other measures will be copied to the authenticated profile. Note the value `apple` is persisted.
- For the eVars that are persisted, no Instance metric is recorded. This means when using cross-device visitor identification, it is possible to see reports where the Unique Visits metric for an eVar value is larger than the Instance metric.

On subsequent data connections

The example below is a representation of how data is sent to Adobe Analytics when a customer authenticates on a new device, after having previously authenticated on a different device:

date_time	visitor id	cust_visid	post_visitor_id	visit_num	visit_page_num	evar16	post_evar16	evar17	post_evar17
2/07/2012 15:33	sv2		5477766334477	1	1				
2/07/2012 15:38	sv2		5477766334477	1	2	pear	pear	train	train
2/07/2012 15:39	sv2	u999	2947539300	2	1				
2/07/2012 15:42	sv2		2947539300	2	2		apple		
2/07/2012 15:42	sv2		2947539300	2	3		apple		

When the customer authenticates his or her id is matched to the previous 'authenticated' profile - 2947539300. The profile used at the start of this visit (5477766334477) is no longer used and no data persists from the file.

- Geo-Segmentation data is recorded based on the first hit of the visit and does not change for a single visit regardless of the device used. This means on a subsequent data connection on a new device, geo-Segmentation data is generally not included.
- Technology columns such as browser, operating system, and color depth are recorded on the first hit of a visit. Like Geo-Segmentation values, they will not be copied to the stitched profile.

- A Marketing Channel such as Direct or Internal which is commonly set up to not overwrite another channel will overwrite other channels on a subsequent data connection containing a first authentication for that device, such as the first authentication shown in [Unique Visitor and Visits Counts](#).

Special cases

In some other cases data is not persisted from the unauthenticated to the authenticated profile.

- If a user is new to your site (has never visited before on this device) AND that user authenticates within approximately 3 minutes of arriving, no values will persist to the authenticated profile.

Visitor Migration

Visitor migration is a process where the visitor ID cookie is migrated from one domain to another.

Visitor migration lets you preserve visitor identification cookies when changing data collection domains. Data collection domains might change for the following reasons:

- Moving from `2o7.net` to `omtrdc.net` ([Regional Data Collection](#)).
- You are implementing the [Experience Cloud Visitor ID Service](#) and are moving from a CNAME/first-party data collection domain to `2o7.net` or `omtrdc.net` ([Regional Data Collection](#)).
- Moving from `2o7.net` or `omtrdc.net` to a cname/first-party data collection ([First-Party Cookies](#)).
- Moving from one CNAME to another (changing domains).

After visitor migration is configured, when a user visits the new domain without a visitor ID cookie, the server redirects to the previous data collection hostname, retrieves any available visitor ID cookies, and then redirects back to the new domain. If a Visitor ID is not found on the previous hostname, a new ID is generated. This occurs only once per visitor.

Visitor Migration Process

The following table lists the tasks required for visitor migration:

Task	Description
<p>To get started: Contact Customer Care with the domain(s) you want to migrate, and the migration period you would like to enable (30, 60, or 90 days). Make sure you include the non-secure and secure domains.</p>	<p>Create a list with the <i>exact</i> syntax for the domains you want to migrate to and migrate from.</p> <ul style="list-style-type: none"> • <code>example.112.2o7.net > metrics.example.com</code> • <code>example.102.112.2o7.net > smetrics.example.com</code> <p>The migration host names are configured on Adobe Data collection server. Customer Care will let you know when the change is made so you can plan for the next step.</p>
<p>6+ hours after configuration change: Update the <code>s.trackingServer</code> and <code>s.trackingServerSecure</code> variables in your Analytics JavaScript code to use the new data collection servers.</p>	<p>After you make this change, use a Packet Analyzer to verify that the Analytics image request is going to the updated data collection server.</p>
<p>Immediately after updating your Analytics code: Test your site to verify</p>	<p>Use a Packet Analyzer to verify that when you access your site for the first time, or after clearing cookies, you see three 302 (redirect) HTTP</p>

Task	Description
that the redirect to the previous data collection domain is occurring.	status codes before the 200 (OK) HTTP status code. If any of these redirects fail, contact Customer Care immediately to ensure that the migration is configured correctly.
For the entire migration period: Keep the DNS record for the previous hostname active.	The previous hostname must resolve through DNS or the cookie migration will not occur.

Deprecated visitorMigrationKey and visitorMigrationServer Variables

As of March 2013, the `visitorMigrationKey`, `visitorMigrationServer`, and `visitorMigrationServerSecure` data collection variables are deprecated and no longer used. The data previously contained in these variables are now stored on Adobe servers for increased security.

Using Timestamps Optional

Learn about the benefits and constraints of using Timestamps Optional setting.

Timestamps Optional is the default setting for all new report suites.

- Mix timestamped and non-timestamped data in the same global report suite.
- Send timestamped data from a mobile app to a global report suite.
- Upgrade apps to employ timestamps without having to create a new report suite.



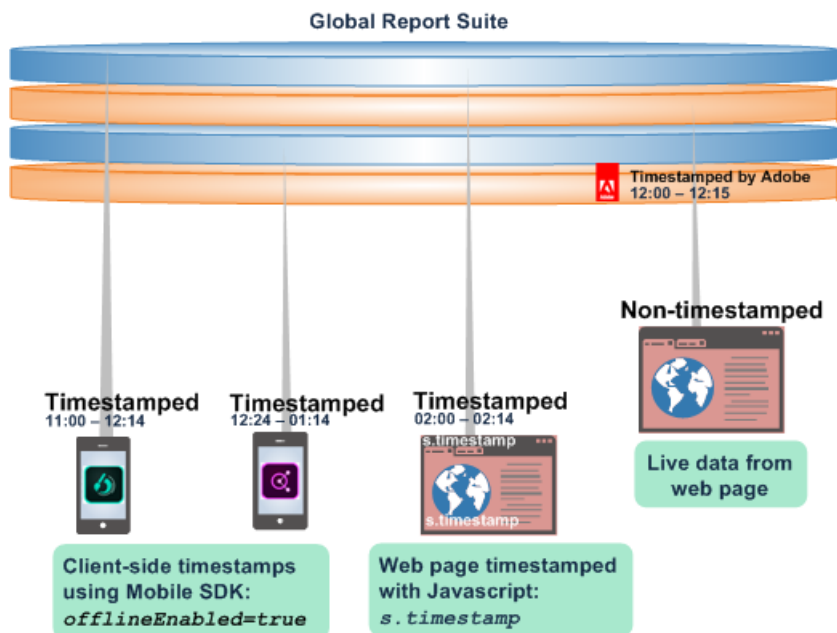
Note: *Timestamps Optional is the default setting for all new report suites generated from a template. New report suites copied from an existing report suite will inherit settings from the original.*

See [Timestamps Optional](#) for additional setup information.

Timestamps Optional: Integrating Timestamped and Non-timestamped Data

Using the Timestamps Optional feature, you can combine non-timestamped data with timestamped data without incurring data loss. Offline data with timestamps generated from a mobile device can be combined with live, non-timestamped data from a web page—or integrated with data from any platform using a client-side timestamp call.

- **Timestamp data.** Client-side timestamp data is captured and sent directly with the device data using client-side timestamp variables: Javascript (`timestamp`) on a web page, or using a Mobile SDK call (`offlineEnabled=true`) in a mobile app.
- **Non-timestamp data.** Adobe sets a timestamp on non-timestamped data in a report suite when the data hits the collection servers.



A report suite can have one of the following timestamp settings:

- Timestamps not allowed (setting visitorID supported)
- Timestamps required (setting visitorID not supported)
- Timestamps optional (setting visitorID supported but not on timestamped hits)

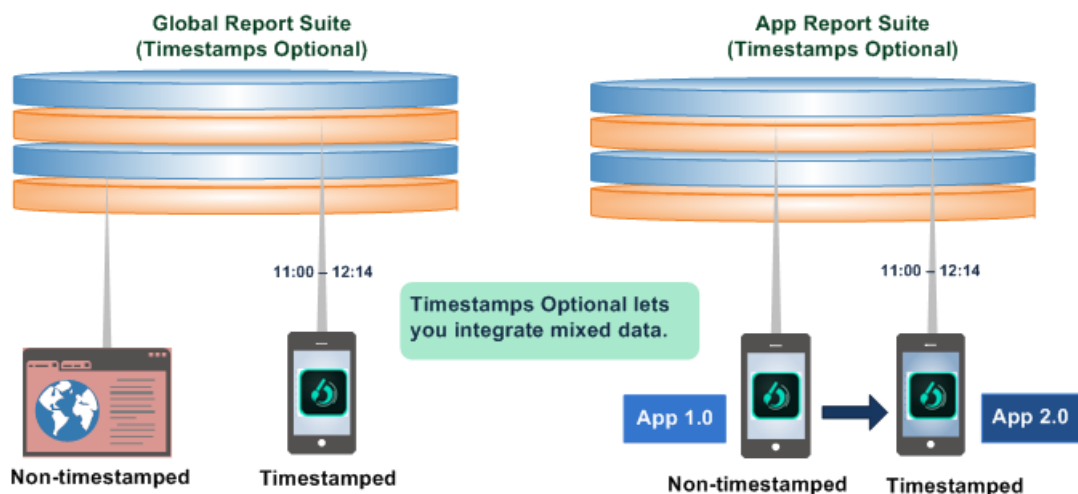
About Timestamps Optional Features

Timestamps Optional allows you to integrate and report across multiple report suites with or without client-side timestamps included. With Timestamps Optional you can update your app to employ timestamps while still using non-timestamped data from the previous app.

In previous releases...	In addition...
<p>Timestamped data could not be sent to a non-timestamped global report suite. Consequently, hit data sent from offline devices was dropped when adding to a non-timestamped report suite.</p>	<p>Updating an app to collect and use timestamps required you to employ a new report suite.</p>

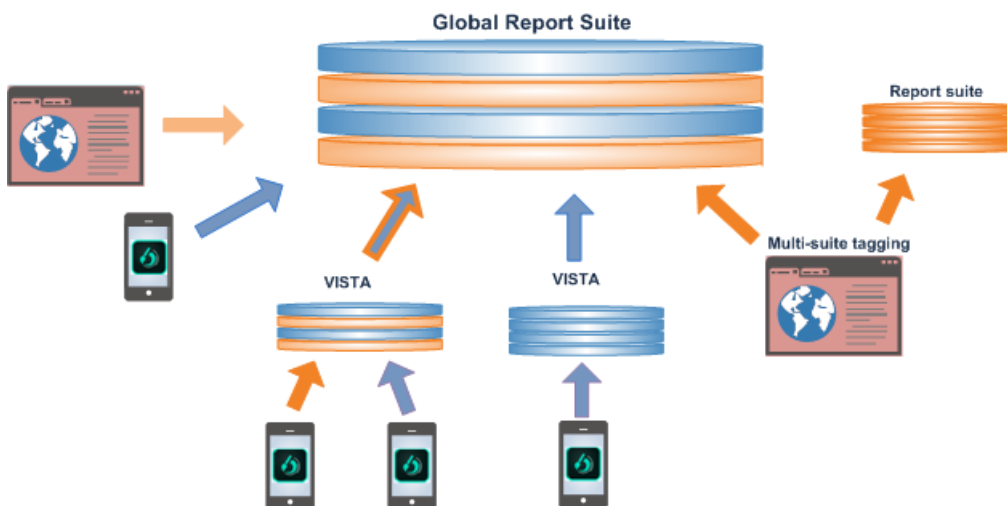
In previous releases...	In addition...
Consequently, hit data sent from offline data was dropped when adding to a non-timestamped report suite.	You could not save to the existing report suite or integrate existing data when updating your app to use timestamps.

With Timestamps Optional, you can integrate non-timestamped data from a live website with offline data from mobile devices, or update your non-timestamped app to a timestamped app.



Combining Data into a Global Report Suite

Combining data into a global report suite can be done in multiple ways, including multi-suite tagging, Vista rules, and imported batch files from offline sources.



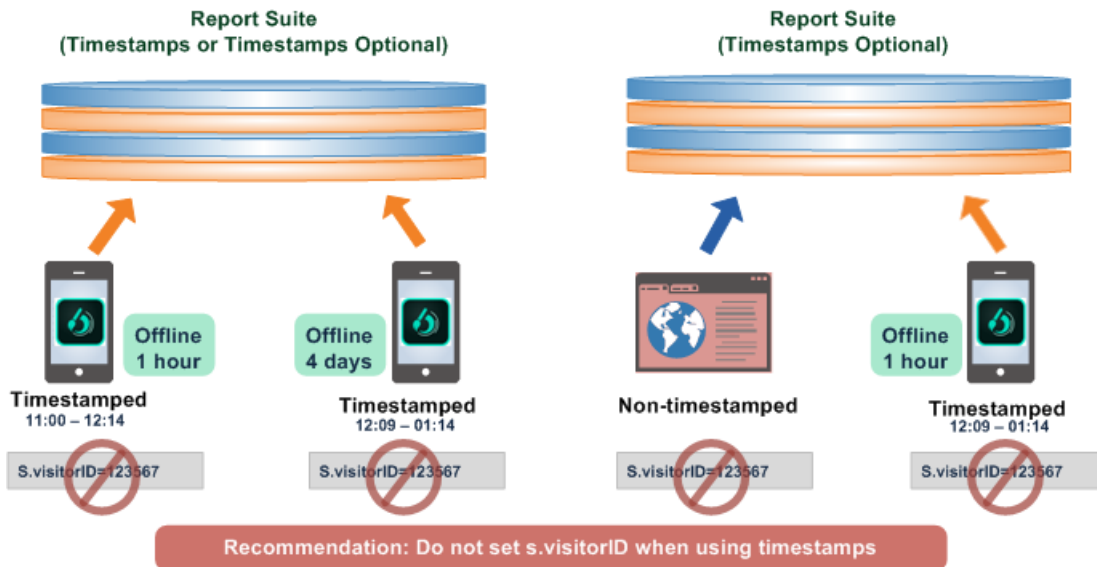
⚠ Important: Carefully plan the design for each component data set so the combination makes sense in a global report suite.

Best Practices when Employing Timestamps

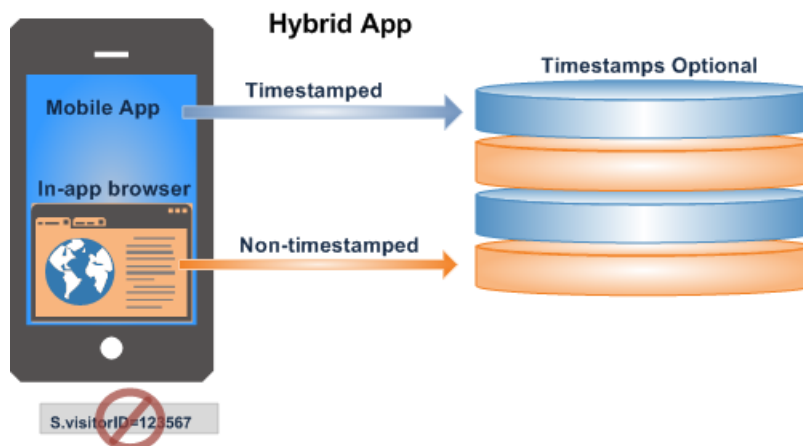
The following are best practices and a few requirements and restrictions to be aware of when integrating timestamped with non-timestamped data.

- In general, timestamps for a given visitor or visit must arrive at Adobe in the correct chronological order.

Out-of-order data can include late arriving data from offline data collection and late arriving hits, or out-of-sync clocks on offline mobile devices. Out-of-order data can negatively impact time calculations (such as time spent values), attribution (eVar persistence), visit number/visit counts, and pathing reports.



- Using timestamps when setting a *s.visitorID* is not recommended. It can lead to out-of-order data.
- Hybrid apps composed of an app (timestamped, offline data) opening a web browser (non-timestamped, live data) should not use timestamps. It results in inaccurate reporting of the session.



In addition, hybrid apps should not set a visitor ID.

Redirects and Aliases

Redirects point the browser to a new location without user interaction. They are executed at either the web browser (client-side redirect) or at the web server (server-side redirect).

Because redirects do not require any user interaction, redirects are often executed without the user ever noticing. The only thing that indicates a redirect has occurred is the browser's address bar. The address bar displays a URL that is different from the link the browser initially requested.

Although there are only two types of redirects, they can be implemented in numerous ways. For example, client-side redirects can occur because the web page to which a user has pointed his or her browser contains scripting or special HTML code that redirects the browser to another URL. Server-side redirects can occur because the page contains server-side scripting or because the web server has been configured to point the user to another URL.

Analytics and Redirects

Analytics gathers some of its data from the browser, and relies upon certain browser properties. Two of those properties, the "Referring URL" (or "referrer") and the "Current URL" can be changed by a server-side redirect. Because the browser is aware that one URL has been requested, but a different URL has been returned, it clears the Referring URL. The result is the referring URL is blank, and Analytics might report that no referrer existed for the page.

The following examples illustrate how browsing is affected without and with redirects:

- [Example: Browsing Without Redirects](#)
- [Example: Browsing With Redirects](#)

Example: Browsing Without Redirects

Consider the following hypothetical scenario in which the user does not encounter a redirect:

1. User points his or her browser to `www.google.com`, and types, "discount airline tickets" into the search field, and then clicks the **Search** button.
2. The browser displays the search results including a link to your site, `http://www.flywithus.com/`. After displaying the search results, the browser's address bar displays the search terms that the user entered in the search field (`http://www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets`). Notice that the search terms are included in the URL query string parameters that follow `http://www.google.com/search?`.
3. The user clicks the link to your hypothetical site `http://www.flywithus.com/`. When the user clicks this link and lands on the `flywithus.com` website, Analytics uses JavaScript to collect the referring URL (`http://www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets`) as well as the current URL (`http://www.flywithus.com/`).
4. Analytics reports the information collected during this interaction in various reports, such as **Referring Domains**, **Search Engines**, and Search Keywords.

Example: Browsing With Redirects

Redirects can cause the browser to blank out the true referring URL. Consider the following scenario:

1. User points his or her browser to `www.google.com`, and types, "discount airline tickets" into the search field, and then clicks the **Search** button.

- The browser window's address bar displays the search terms that the user typed into the search field (`http://www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets`). Notice that the search terms are included in the URL query string parameters that follow "`http://www.google.com/search?`". The browser also displays a page that contains the search results including a link to one of your domain names: `http://www.flytohawaiiiforfree.com/`. This *vanity* domain is configured to redirect the user to `http://www.flywithus.com/`.
- The user clicks on the link `http://www.flytohawaiiiforfree.com/` and is redirected by the server to your main site, `http://www.flywithus.com`. When the redirection occurs, the data that is important to Analytics data collection is lost because the browser clears the referring URL. Thus, the original search information used in the Analytics reports (for example, **Referring Domains**, **Search Engines**, **Search Keywords**) is lost.

[Implementing Redirects](#) discusses how to leverage Analytics variables to capture the data lost in the redirect. Specifically, the section discusses how to fix the "discount airline tickets" situation described above.

Implementing Redirects

In order to capture Analytics data from redirects, four minor alterations need to be made to the code that creates the redirect and the AppMeasurement for JavaScript file.

Completing the following steps will retain the information that the original referrer (for example, `http://www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets` in the scenario above) passes to your site:

Configure Referrer Override JavaScript Code

The code snippet below shows two JavaScript variables, `s_referrer` and `s_pageURL`. This code is placed on the ultimate landing page of the redirect.

```
<script language="JavaScript"
src="//INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/AppMeasurement.js"></script>
<script language="JavaScript"><!--
/* You may give each page an identifying name, server, and channel on
the next lines. */
s.pageName=""
s.server=""
s.campaign=""
s.referrer=""
s.pageURL=""
```



Important: Set `s.referrer` only once on the page. Setting it more than once with every tracking call or with every link click that is tracked causes double counting of the referrer and related dimensions, such as search engines and keywords.

Redirects using `getQueryParam`

While the `getQueryParam` is an easy way to populate Analytics variables with query string values, it must be implemented in connection with a temporary variable so that legitimate referrers are not overwritten when the query string is empty. The best way to use `getQueryParam` is in connection with the `getValue` plug in as outlined with the following pseudo-code.

```
// AppMeasurement 1.x
var tempVar=s.Util.getQueryParam('origref')
if(tempVar)
```

```
s.referrer=tempVar;

// H Code
var tempVar=s.getQueryParam('origref')
if(tempVar)
  s.referrer=tempVar;
```

Modify the Redirect Mechanism

Modify the Redirect Mechanism XE "Redirect Mechanism"

Because the browser strips referring URL, you must configure the mechanism that handles the redirect (for example, the web server, server-side code, client-side code) to pass along the original referrer information. If you would also like to record the alias link URL, this must also be passed along to the ultimate landing page. Use the `s_pageURL` variable to override the current URL.

Because there are many ways to implement a redirect, you should check with your web operations group or your online advertising partner to identify the specific mechanisms that execute redirects on your website.

Capture the Original Referrer

Capture the Original Referrer XE "Original Referrer"

Normally, Analytics will obtain the referring URL from the browser's `document.referrer` property, and the current URL from the `document.location` property. By passing values to the `referrer` and `pageURL` variables, you can override the default processing. By passing a value to the `referrer` variable, you are telling Analytics to ignore the referrer information in the `document.referrer` property and to use an alternative value that you define.

Therefore, the final version of the landing page would need to contain the following code to correct the issues introduced in the "discount airline tickets" scenario.

```
<script language="JavaScript"
src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/AppMeasurement.js"></script>
<script language="JavaScript"><!--
/* You may give each page an identifying name, server, and channel on
the next lines. */
s.pageName=""
s.server=""
s.campaign=""
s.referrer="http://www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets"
// Setting the s.pageURL variable is optional.
s.pageURL="http://www.flytohawaiiiforfree.com"
```

Verify the Referrer with the Adobe Debugger

Run a test to verify that the referrer, originating URL (`s_server`) and campaign variables are being captured.

These variables will be represented as the following parameters in the [Adobe Debugger](#).

	URL or Query String Value	Value as Shown in the DigitalPulse Debugger
Original Referrer	http://www.google.com/search%3Fhl%3Den%26ie%3DUTF826q%3Ddiscount%2Bairline%2Btickets	r=http://ref=www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets
Page URL	http://www.flytohawaiiiforfree.com	g=http://www.flytohawaiiiforfree.com

	URL or Query String Value	Value as Shown in the DigitalPulse Debugger
		This value will appear in the DigitalPulse Debugger if the <i>pageURL</i> variable is used.
Ultimate Landing Page URL	<code>http://www.flywithus.com</code>	This value will NOT appear in the DigitalPulse Debugger if the <i>pageURL</i> variable is used.

The text that the debugger displays should correspond to the following example:

Image

```
http://flywithuscom.112.2o7.net/b/ss/flywithuscom/1/JS-1.4/s61944015791667?[AQB]
ndh=1
t=4/8/2014 13:34:28 4 360
pageName=Welcome to FlyWithUs.com
r=http://ref=www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets
cc=USD
g=http://www.flytohawaiiiforfree.com
s=1280x1024
c=32
j=1.3
v=Y
k=Y
bw=1029
bh=716
ct=lan
hp=N
[AQE]
```

After verifying that the Adobe Debugger displays these variables, it is always helpful to confirm that the search terms and the original referring domain (prior to the redirect) are registering traffic in reports.

Testing and Validation

Testing and Validation Process

Use validation and testing to ensure data reporting accuracy.

Validation and testing should always be done on the development report suite.

Identifying the `s_account` Variable in the DigitalPulse Debugger

When you run the **DigitalPulse Debugger**, you may want to look for the `s_account` variable.

The following figure shows the location of the `s_account` variable.

Image

```
http://omniture.112.2o7.net/b/ss/omnicom/1/G.9p2/s9569
8397722543?[AQB]
ndh=1
t=12/9/2006 11:2:17 4 360
pageName=Homepage
g=http://www.omniture.com/
cc=USD
c1=Homepage
c2=Homepage
v9=Homepage
v15=general
[AQE]
```

JavaScript JS File

Verify that the `.JS` file is correctly referenced from the page. The path can be specified either relative to the current document, or an absolute path name can be used.

```
<script language="JavaScript"
src="http://www.sampleco.com/javascript/includes/s_code.js"></script>
```

If some pages of the site are loaded in a secure protocol (`https:`), and reference the `AppMeasurement` for JavaScript file, ensure that the reference to the file is either secure (via `https:`) or code the reference as shown below. This example adopts the protocol of the current page and prevents the warning that "some elements are non-secure."

```
<script language="JavaScript"
src="//www.sampleco.com/javascript/includes/s_code.js"></script>
```

Ensure that the `.JS` file on the web servers have permissions appropriately set so that the file may be downloaded and executed by website visitors. If a different `.JS` file is used on development servers, set the "read only" attribute for the `.JS` file on production servers to avoid an overwrite. If altered, ensure that the following settings are set appropriately at the top of the `.JS` file:

```
/* ***** CONFIG SECTION ***** */
/* You may add or alter any code config here. */
```

```

/* Link Tracking Config */
s.trackDownloadLinks=false /* true for download tracking */
s.trackExternalLinks=false /* true for exit link tracking */
s.trackInlineStats=false /* true for ClickMap support */
s.linkDownloadFileTypes="exe,zip,wav,mp3,mov,mpg,avi,doc,pdf,xls"
s.linkInternalFilters="javascript:"
s.linkLeaveQueryString=false
s.linkTrackVars="None"
s.linkTrackEvents="None"

```

If "*s_account*" is assigned a value at the top of the .JS file, ensure that the report suite ID (populated in the *s_account* variable) is correct. Also ensure that the code in the page is not setting the **Report Suite ID** (*s_account* variable).

Examine the image request and variables to ensure that the "fallback method" (the third part of the "split" code in the example above) is not creating the image request instead of the .JS file. This can be determined since the "fallback" method only creates an image request with minimal information.

Code Modifications

Testing any modifications to the .JS file or HTML code is the responsibility of the customer. It should be completed prior to publishing the modifications to production websites.

Ensure that the linefeeds/return characters are not stripped or altered from the code that is placed within the HTML, or from within the .JS file. Ensure that the JavaScript executes without an error on all pages and page templates (in Internet Explorer Internet Options, select the Advanced Tab, and click Display a Notification about Every Script Error).

When testing for errors, paste the code into a default HTML page to determine if the error is occurring because of other page elements/objects.

```

<html><head></head><body>
..paste code here to debug...
</body></html>

```

Variables and Values

Ensure that the variables that are populated from server scripting or code cannot output any quotation marks that interfere with the values.

For instance:

```

s.pageName="Article: "Article Name" "
s.pageName='Company's Information'

```

Ensure that the values of the variables do not exceed their maximum limits, specified in this document. Additional characters are cropped at the data collection servers. They may interfere with the total length, increase bandwidth unnecessarily, and may cause other issues.

Do not use \$, [™], ®, ©, or commas (,) in the products variable. Generally, these are not useful in any Analytics variables and may interfere with the ability to interpret or export fields. The best practice is to limit characters to the first 127 ASCII characters.

Ensure that the events variable is populated with an appropriate value (**prodView**, **purchase**, **scAdd**, **scRemove**, **scOpen**, or event1-event5) whenever *products* is populated. Ensure that the case of all Analytics variables and functions are maintained, as shown below.

```

s.pageName
s.server
s.channel
s.pageType
s.prop1 - s.prop20

```

```
s.campaign
s.state
s.zip
s.events
s.products
s.purchaseID
s.eVar1 - s.eVar20
var s_code=s.t();if(s_code)document.write(s_code)//-->
```

Page names are case sensitive, and differences create additional page records. "Home" and "home" are two different pages within Analytics.



Note: Multiple page records cannot be combined within reports.

Validate that links are reported in the **Custom Links** report. Ensure that the correct parameters are passed to the **tl** function. For more information on **custom links**, see [The s.tl\(\) Function - Link Tracking](#).

Custom Variables

List of **custom** variables used in Analytics.

Variable	Description
Traffic Variables	<p>Check the value of prop1 - 75. Here is a checklist of items to check.</p> <ul style="list-style-type: none"> • Is the correct case used? "ValueA" is a different record than "valueA." You can use all lowercase since a small subset of browsers convert all variables to lower case. • Are the values less than 100 characters in length? If not, some clipping of the values can occur. • Are all the values in a single property variable related, or do some values look out of place?
Conversion Variables	<p>Econversion variables include eVar 1 - 75. Here is a list of issues to check for the following.</p> <ul style="list-style-type: none"> • Is the correct case used? "ValueA" is a different record than "valueA." You can use all lowercase since a small subset of browsers convert all variables to lower case. • Are the values less than 255 characters in length? If not, some clipping of the values can occur. • Are all the values in a single eVar related, or do some values look out of place?
Custom Events	<p>Events include both standard values (prodView, scOpen, scAdd, scCheckout, purchase), as well as custom events from event1 to event100. All events are sent in the events variable. Multiple events on the same page should be comma-delimited (no white space).</p> <ul style="list-style-type: none"> • For all the standard conversion events, products should also be populated with the applicable products. For all events except purchase, the qty and price elements are optional. • The purchase event, must be set only once in a session after the purchase has been completed and confirmed.

Implementation Acceptance

Implementation process steps.

The following steps outline the implementation process.

1. The Adobe Consultant gathers report requirements and creates a data collection plan based on those requirements.
The data collection plan includes variable definitions, required VISTA rules and custom JavaScript, data correlation, and all settings for each report suite. The client completes the Implementation Questionnaire.
2. Technical resources on the client-side implement the code, site-specific JavaScript, and server-side variables.
3. The Adobe Consultant addresses technical issues during the implementation and assists in devising solutions as required.
4. Technical resources on the client-side unit test the implementation.
Testers log in to Analytics and verifying all variables (*page name, channel, server, events, campaign, econversion variables, custom traffic variables, products, and all other variables*).
5. The client notifies Adobe that the implementation is complete.
The client provides a validation sample (data sample) to the Adobe Consultant to validate data accuracy. (VISTA-generated report suites are validated by comparing appropriate metrics. A client-Adobe agreement of the metrics to be validated for such report suites shall be made in advance, at the time of the VISTA rule creation.)
6. The client faxes (or signs online) an Implementation Acceptance and Agreement for the appropriate site(s).
7. After the acceptance has been received, the Adobe Consultant enables the Adobe Best Practices - Implementation Verification certification within the interface.
8. Optionally, the client can contract with Adobe for monitoring services for key pages of the implemented site (generally, these are the primary templates, home page, and critical entry pages).
This monitoring software is described in a separate document, but tracks pages by loading and executing the page, then comparing the image request to a baseline stored in a database. If any differences are detected, the software notifies specified Adobe (AM/IE) and client personnel via email.

The following items help to ensure a successful implementation:

- A best practices document, which is client-facing and explains the processes in detail.
- The validation document that the customer uses to unit test the implementation.
- An Implementation Acceptance and Agreement form for the client to sign.
- A monitoring application that continuously validates the tags.
- A relationship with Accenture to help with implementation testing.
- Utilities and/or tools for comparison of page views and/or orders. Those comparisons can get fairly difficult.
- A method or process to quickly obtain the debug log for a given day, by report suite ID.

Data Accuracy Validation

Data accuracy validation is a process of comparing report data with known and verifiable data points.

The validation process should be completed by Adobe personnel, preferably by the Adobe Consultant (the person most familiar with the technical implementation details).

The preferred data points for this validation, in order of preference, are listed as follows:

- (Econversion sites) Comparison of econversion orders for a single day.
- Comparison of known success events, especially logged data where IP address and other browser information generally stored in web server logs can be compared to the data collected.
- Comparison of page views.



Note: Default pages, such as *index.html*, often receive automated or monitoring traffic. These pages represent a greater difference to browser-based data collection than other visited pages.

All three types of validation require a debug log or data feed for the time period in question. This is generally one day or less.

It is expected that orders or success events can be measured to within 2-3% of actual values (sometimes reaching higher accuracy levels) using standard JavaScript-based implementations. This assumes an SSL page, since SSL pages are cached much less frequently, and by definition they should not be cached. An implementation with fully server-side image requests on an SSL page should come within about 0-1% of actual values. Non-secure pages may experience higher differences, but still within 5% of actual values.

When comparing page views for a single time period, it is expected that the page views can be accounted for within 5% of actual values, not including monitoring (such as Keynote or WhatsUpGold) or automated traffic (spiders, bots, and scripts).

Data accuracy comparisons need to take into account the following items:

- QA or other types of internal testing that may be filtered by IP addresses or VISTA rules.
- Smart tags that only generate tags for certain types of orders or traffic.
- Queries for comparison must take into account what is being measured by the website (not including returns, orders placed by customer service personnel, or other special conditions).
- Ensure that the time zone differences between the query and the report suite match.
- Custom Keynote or similar traffic (Keynote Transaction, etc.) that measure the ordering process and may be reflected in tags, but removed from ordering systems.
- Account for the client's de-duping processes.
- Reloads of the order page (Orders are de-duplicated based on *purchaseID*).

Experience Cloud Debugger

Install the Adobe Experience Cloud Debugger. The debugger inspects tags for the Analytics Cloud, Adobe Target, Advertising Cloud, Experience Cloud ID Service, Dynamic Tag Management, and Launch, by Adobe.



Important: *This JavaScript bookmarklet is no longer maintained. Adobe recommends using the [Adobe Experience Cloud Debugger](#) extension for Chrome.*

When executed in your browser, it shows the image requests that transmitted data from that page into Experience Cloud solutions, along with any variable values or parameters that were captured. This allows you and your developers to check the validity of your implementation on any page on your site.

The DigitalPulse Debugger is officially supported for use in all recent versions and builds of Mozilla Firefox, Google Chrome, Microsoft Internet Explorer, and Safari.



Note: *Because the Debugger functions by creating a pop-up window when you access a special bookmark in your web browser, certain ad-blocking plug-ins and pop-up blockers might interfere with the loading of the Debugger.*

Install the debugger in Chrome

Install the Experience Cloud Debugger extension in the Google Chrome browser.

1. Navigate to [Adobe Experience Cloud Debugger](#).
2. Follow the instructions to add the extension to Chrome.


Install the debugger in Firefox (not supported)

How to create a bookmark for the Adobe Debugger from within Mozilla Firefox.

1. Right-click the bookmarks sidebar, then click **New Bookmark**.
2. In the **Name** field, specify "Adobe Debugger" as the name for the new bookmark.
3. In the **Location** field, paste the code that you copied to your clipboard as explained in [Adobe Debugger Installation](#).
4. Select **Load the Bookmark in the Sidebar**, if desired.
5. Click **Add**.

Install the debugger in Internet Explorer (not supported)

How to create a bookmark for the Adobe Debugger from within Internet explorer.

1. In the **Favorites Bar**, click the **Add to Favorites Bar** icon ( icon).
If the **Favorites Bar** is hidden, right-click the browser header, then click **Favorites Bar**.
A new bookmark is created.
2. Right-click the bookmark, then click **Rename**.
3. In the **New Name** field, specify "Adobe Debugger" as the name, then click **OK**.
4. Right-click the newly created bookmark again, then click **Properties**.
5. In the **URL** field, paste the code that you copied to your clipboard as explained in [Adobe Debugger Installation](#).
6. Click **OK**.

Packet Analyzers

Packet analyzers let you view the data sent by your implementation to Adobe Data Collection Servers.

Similar to the DigitalPulse Debugger, a packet monitor shows what data parameters are being passed in an image request; however, packet monitors provide added functionality:

- View custom link tracking image requests
- View image requests using implementation methods other than JavaScript, such as hard-coded image requests or Appmeasurement

To view Analytics requests, filter outgoing requests using "b/ss".


In very rare cases, the debugger will report an image request although no request makes it to Adobe's Analytics processing servers. Using a packet monitor is a great way to be 100% sure that a specific image request is being fired successfully.

While Adobe does not provide an official packet monitor, there are a wide range of them on the internet. The following are some packet monitors others have found useful.



Note: *These lists are not meant to be comprehensive, but rather information on frequently used monitors. If you have a packet monitor you successfully use and find useful, feel free to provide feedback using the **Feedback** button on the right side of this window.*

Firefox	Internet Explorer	Chrome	Standalone Programs
Observe Point (tag viewer)	HttpWatch	Observe Point (tag viewer)	Charles
HttpFox		Chrome Developer Tools	Fiddler
Tamper Data		Firebug Lite	Wireshark
HttpWatch			
Firebug			

 **Note:** Adobe does NOT support or troubleshoot any issues you may experience with these packet monitors. Consult the packet monitor's originating site for assistance instead.

NS_Binding_Aborted in Packet Monitors

The **NS_BINDING_ABORTED** message is often seen on **custom link tracking** image requests with packet sniffers that sit on top of the browser, such as Tamper Data or HTTPFox .

This error occurs because the link tracking image request is designed to let the browser proceed to the next page before waiting for a response from the Adobe data collection servers.

Adobe's response to the image request is simply a blank 1x1 transparent image, which is not relevant to the content of the page. If you see a line item in your packet monitor from Adobe, either with a **200 OK** response or an **NS_BINDING_ABORTED** response, the data has reached our servers. There is no need to have the page wait any longer.

Packet monitors integrated as a plug-in rarely see the full response. They tend to see the request as aborted because the full response was not received. These monitors also rarely make a distinction between whether it was the request or response that was aborted. A stand alone packet monitor typically has more detailed messages and reports the status more accurately. For example, a user may get a message in *Charles* saying "Client closed connection before receiving entire response." This means the data did reach our servers, just the browser moved on to the next page before the 1x1 pixel was received.


If an external packet sniffer is reporting that the data collection request is aborted, rather than the response, this is a cause for concern. Adobe Customer Care can provide help in troubleshooting.

Common Syntax Mistakes

A successful implementation means drawing useful data from your website. An unsuccessful implementation can mean spending time looking through your JavaScript code to find errors. In an attempt to save you time and prevent frustration, some of the most common code mistakes users make when implementing are listed in the following sections.

Putting Analytics Code in the Head Tag

Analytics code creates an image object, a non-visible image that does not show up on your page.

 **Note:** This section applies only to the legacy `s_code.js` implementation. [About AppMeasurement for JavaScript](#) supports deploying the library and page code in the `<head>` tag.

Previously, a common implementation practice was to place the Analytics JavaScript code between the <head> and </head> tags. By placing the code between these tags it prevented the 1 x 1 pixel image that was returned by the request that sent data into Adobe servers from affecting page layout in any way. Putting code in the document head means the code appears earlier in the code. This lets it execute sooner, which lets you count page views for partial page loads more effectively.

Certain elements of the code require the existence of the body object. Since Web browsers execute code in the order they receive it, if the Analytics JavaScript code is in the document head, it executes before the body object exists. As a result, your implementation does not collect **ClickMap** data, and automatic tracking of file downloads or **exit** links are not available. You also do not receive connection type data or visitor home page data. Putting the code in the document head works, but the result is a very limited version of Analytics, and users may wonder why certain reports and tools, including **ClickMap**, aren't recording data.

The Analytics code can be placed anywhere inside BODY tags (<BODY></BODY>) of a well-formed HTML page. Adobe recommends placing the code in a global include file at the top of the page (inside the HTML body tag). The code can be placed anywhere on the page, except as noted below:

- If placed within a table, post the code only within the <td></td> tags. For example, do not place the code between an opening <tr> tag and an opening <td> tag.
- The code that sets the variables must occur after the reference to the `s_code.js` file.
- Make certain that the **report suite IDs** in the `s_account` variable in the `s_code.js` file are set correctly. This variable is typically set correctly when downloading code from the Code Manager for a particular report suite, or as supplied by an Adobe Technical Consultant.

If you want to integrate Analytics with Target, the JavaScript include file must be placed at the bottom of the page. The following example shows correct placement of the Analytics code:

```
<html>
<head></head>
<body>
<!-- Analytics code version: H.20.3.
Copyright 1997-2009 Omniture, Inc. More info available at
http://www.omniture.com -->
<script language="JavaScript" type="text/javascript"
src="http://www.yourdomain.com/js/s_code.js"></script>
<script language="JavaScript" type="text/javascript"><!--
/* You may give each page an identifying name, server, and channel on
the next lines. */
s.pageName=""
s.server=""
s.channel=""
s.pageType=""
s.prop1=""
s.prop2=""
s.prop3=""
s.prop4=""
s.prop5=""
/* Conversion Variables */
s.campaign=""
s.state=""
s.zip=""
s.events=""
s.products=""
s.purchaseID=""
s.eVar1=""
s.eVar2=""
s.eVar3=""
s.eVar4=""
s.eVar5=""
/***** DO NOT ALTER ANYTHING BELOW THIS LINE ! *****/
var s_code=s.t();if(s_code)document.write(s_code)//--></script>
<!-- End Analytics code version: H.20.3. -->
```

```
</body>
</html>
```

Using `s.linkTrackVars` and `s.linkTrackEvents`

The key to a successful link tracking implementation is understanding the `s.linkTrackVars` and `s.linkTrackEvents` variables. This lets you pass custom variable values on user actions.

If you are implementing custom link tracking and are setting **custom** variables and *events*, make sure that your `s.linkTrackVars` variable contains a comma-separated list of all variables that you are passing, including the *events* variable. Make sure that `s.linkTrackEvents` includes a comma-separated list of all events that you are passing.

Setting `s.linkTrackVars` and `s.linkTrackEvents` does not actually set these variables/events, it only prepares the Analytics code to do so. You still need to set the variables manually, as shown in the example below:

```
<script language="javascript">
function customLinks () {
  var s=s_gi('myreportsuite');
  s.linkTrackVars="prop1,eVar7,products,events";
  s.linkTrackEvents="event5,event9";
  s.prop1="Link Click";
  s.eVar7="my_page.html";
  s.events="event5";
  s.tl(true,'o','Custom Link Click');
}
</script>
<a href="my_page.html" onclick="customLinks();">My Page</a>
```

Notice that *events* is listed in the `s.linkTrackVars` variable. The individual events that may be passed are included in the `s.linkTrackEvents` variable and are also included within `s.events` later in the function. Each of the variables that are passed are listed in `s.linkTrackVars` before they are populated later in the function. Also, "event9" has been included in `s.linkTrackEvents`, but has not been included in `s.events`. It is not recorded, but could be recorded if the user had chosen to set `s.events="event5,event9"`.

Automatic file download and **Exit** link tracking work differently. The `s.linkTrackVars` and `s.linkTrackEvents` are included in the standard `s_code.js` file, and both are set to none. These variables are typically left set to none in the `s_code.js` file so that automatic link tracking, unlike **custom link tracking**, uses the `s.linkTrackVars` and `s.linkTrackEvents` values that you set in the global JavaScript file. They also pass whatever the existing values of those variables are whenever a file download or **Exit** link is recorded.

Consider the example where `s.channel="Home"` when the page loads, and where `s.linkTrackVars="channel"` in your `s_code.js` file. If a user clicks to download a file, automatic file download tracking passes data into Analytics, including the value of `s.channel` that was set on page load. "Home" is passed a second time, leading to inflation in page view data for this value in the **Site Sections** report.

Adobe strongly recommends leaving the `s.linkTrackVars` and `s.linkTrackEvents` set to "none" in the global JavaScript file and setting them explicitly as necessary with your **custom link tracking** implementation.

Common Mistakes in the Products Variable

The `s.products` variable may be the most syntactically complex variable used by data collection.

Commas, semi-colons, pipes, and equals signs all play specific roles in the variable. It has no overall maximum length, but each individual product entry cannot be longer than 100 bytes (including multi-byte characters). Mistakes in implementation of this variable are understandable, but unfortunately for developers, `s.products` is often a site's most important variable because it makes possible the tracking of revenue, units, product names, and so forth.

Here are a few extremely easy-to-make mistakes that can cause issues on any implementation.

Make sure that your **category**, **product name**, and **revenue** totals are devoid of commas and semi-colons. The comma is used to separate entries in the **s.products** string. This happens when you have two products in the same transaction, the semi-colon is used to delimit fields within an entry. If you use a comma or semi-colon in any other way, data collection assumes that you are separating product entries. Consider the following example:

```
s.products="widgets;large widget, 40 x40 ;1;19.99,wugs;tiny wug;2;1,999.98";
```

In this implementation, the developer probably intended for data collection to read this as shown below:

Category 1: widgets

Product 1: large widget, 40 x40

Units 1: 1

Revenue 1: 19.99

Category 2: wugs

Product 2: tiny wug

Units 2: 2

Revenue 2: 1,999.98

Note the commas in the Product 1 and Revenue 2 entries. These indicate a new product entry. Data collection would interpret the above as:

Category 1: widgets

Product 1: large widget

Category 2: 40'x40'

Product 2: 1

Units 2: 19.99

Category 3: wugs

Product 3: tiny wug

Units 3: 2

Revenue 3: 1

Category 4: 999.98

A mistake like this often results in unexpected numerical values in the **Products** report, because the units field is recorded as the product name. If you see invalid product names in your **Products** report, review your **s.products** variable implementation for misuse of reserved characters, like the comma.

Your product and category names should not contain unsupported characters. This can be especially difficult in the **s.products** string, because product names are often likely to contain characters such as TM, ©, and ®. These characters need to be stripped out of the product and category values before they are placed into **s.products**. You also need to ensure that currency symbols are not included in your revenue values. Supported characters are numbers 1-127 from the ASCII table.

If you are not passing a product category in the product string, make sure to include a semi-colon (;) where the product category is normally displayed, as shown below:

```
s.products=";product name"
```

In this case, the semi-colon represents a placeholder for the product category. If the semi-colon is left out of the product string, then "product name" would be counted as the category, the number of units to be counted as the product name, the revenue to be counted as the units, and so on.

Setting the PageType Variable Correctly

The *pageType* variable is used only to designate a 404 (Page Not Found) error page.

It has only one possible value, which is `errorPage`.

```
pageType="errorPage"
```

On a 404 error page, the *pageName* variable should not be populated. The *pageType* variable should be set only on a designated 404 error page. Any page containing content should never have a value in the *pageType* variable. For pages containing content, you can set the variable as shown below:

```
pageType=" "
```

It is best to delete the variable completely from pages containing content. This practice is recommended to avoid confusion regarding the purpose of the variable.

Using White Space in Variable Values

In HTML there are several characters that create whitespace.

These include a space, a tab, and a carriage return (or linefeed). Consider the following example:

```
<head>
  <title>
    Home Page
  </title>
</head>
<body>
<script language="javascript">
  s.pageName=document.title
</script>
```

In this case, `document.title` populates **s.pageName**, which should receive a value of "Home Page." Notice the space before "Home Page." Not all browsers interpret this white space in the same way. The result may be either of two examples below:

```
s.pageName="Home Page"
```

```
s.pageName=" Home Page"
```

The first value displays correctly, but the second displays white space before the text. Analytics treats these as distinct values for the **s.pageName** variable. The Analytics interface strips the leading white space from the second value. The result is a report that displays as shown below.

Page	Page Views	
1. Home Page	10	76.9%
2. Home Page	3	23.1%
Total		13

This implementation error causes your variable values to be fragmented across multiple line items. SAINT does not allow leading white space in a key value. This means that it cannot be used to group multiple line items as a

work-around if this issue is affecting your site. The only way to fix the problem is to pre-process the desired variable value (in this case, the `document.title` property) to remove any leading (or trailing) white space.

The example above uses the `s.pageName` variable with the `document.title` property. Adobe does not recommend using `document.title` as the page name, nor does this issue only affect the `s.pageName` variable. Any variable that has leading/trailing white space in its value can be affected.

Using Quotes

When you input values into a variable, there are a few best practices to follow.

You can use single quotes or double quotes, make sure you are consistent. If you are using single quotes, always use single quotes. If you are using double quotes, always use double quotes. Both of the examples below are correct.

```
s.prop2='test' (single quotes)
```

```
s.prop2="test" (double quotes)
```

Make sure that you have smart quotes turned off. If you are using single quotes, and you have an apostrophe in the variable value, JavaScript interprets the apostrophe as a single quote. This means it is the end of the string. Consider the following example:

```
s.pageName='John's Home Page'
```

In this case, JavaScript would interpret the apostrophe (which is also a single quote) in "John's" to be the end of the string. Since "s Home Page" has no meaning in JavaScript, it causes an error that prevents the image request from occurring. Either of the following examples would work:

```
s.pageName='John\'s Home Page'
```

```
s.pageName="John's Home Page"
```

In the first example, you can escape the apostrophe by inserting a backslash (\) immediately before it. This tells JavaScript that the apostrophe is not ending the string, but rather is part of it. The string then ends as intended, at the closing single-quote. The second example uses double-quotes around the entire string. In this case, a single-quote cannot indicate the end of the string. The same is true if a double-quote is part of the string. A value of `s.pageName="Team Says "We Win!"` would be incorrect because of the use of double-quotes within the string, as well as surrounding it. This would be correct if entered as `s.pageName="Team Says \"We Win!\""`.

Replacing Your Analytics Code

Adobe offers some best practices for updating your Analytics code.

Frequently, customers use the Adobe **Code Manager** to replace their code with the most recent version. This practice is good to follow as long as you keep certain things in mind.

Using the Incorrect Data Collection Server ID

Occasionally, generic `s_code.js` files that have not been generated from Adobe Code Manager are sent to those implementing the code on your site. As a result, the incorrect data collection server ID (as shown in the `s.dc` variable) for the account is used. It is best to generate new code directly from the **Code Manager** for a specific report suite, rather than reusing code from a different report suite. This is the best way to make sure the `s.dc` variable is populated correctly.

Plug-ins

Some customers implement plug-ins to enhance their Adobe experience. When you replace your code, do not forget that you have plug-ins as part of that code. The code created in the **Code Manager** does not have any plug-in code with it, so all plug-ins need to migrate manually to the newer code base. Make a copy of your existing code just in case something happens, and you need to revert to your previous code.

Table of Common Syntax Mistakes

The following table shows the difference between correct and incorrect code mistakes.

Incorrect	Correct
prop1	s.prop1 (uses "s.")
s.evar1	s.eVar1 (uses correct capitalization)
s.pagetype='errorpage';	s.pageType='errorPage'; (uses correct capitalization)
s.tl(this,o,pageA)	s.tl(this,'o','pageA'); (correct usage of single quotes)
s.events='event1'; s.events='event2';	s.events='event1,event2'; (correct format)
s.pageName="John" (smart quotes on)	s.pageName="John" (smart quotes off)
s.products="shoes,UX4879,1,19.99"	s.products="shoes;UX4879;1;19.99" (uses semicolons, not commas)
s.products=";MacBook Air;1;\$1999.99"	s.products=";MacBook Air;1;1999.99" (does not use dollar signs)
s.products=";Nikon SB-600 Speedlight Flash for Nikon Digital SLR Cameras;1;229.9489183"	s.products=";Nikon SB-600 Speedlight Flash for Nikon Digital SLR Cameras;1;229.95" (round or truncate long prices)
var s_account="rsid1, rsid2"	var s_account="rsid1,rsid2" (no space between report suite IDs when doing multi-suite tagging)
s.events="event1, event2"	s.events="event1,event2" (no space between event IDs when doing multi-event tagging)
s.products="product name"	s.products=";product name" (semicolon used when no product category is listed)

Additional Notes

Keep the closing HTML comment at the very end of the Adobe code (even if you are using the NOSCRIPT part of the script).

Vulnerability Scanner

Whenever users are allowed to input values that are stored in Analytics or any other JavaScript variables, values should be properly escaped or URIEncoded.

We highly recommend using a vulnerability scanning service to avoid injection attacks and other potential exploits that might be present in your code.

Optimizing your Implementation

Analytics deployment is organized into three major steps.

1. This involves pasting a snippet of HTML code onto each page (or page template) of a website. The HTML code snippet is very small (400 to 1,000 bytes) and contains JavaScript variables and other identifiers that facilitate the data collection process.
2. The code snippet calls a JavaScript library file, which contains JavaScript functions specific to Analytics used during metrics collection. If the Analytics code is implemented correctly, the time required for the browser to execute the JavaScript library file is usually negligible.
3. The library file makes an image request to an Adobe data collection server. The server collects the data being submitted and returns a 1x1 transparent image to the visitor's browser. This third step adds an insignificant increment to the total page download time.



Note: Customers can take additional steps to minimize Analytics overhead.

Page Naming

The `pageName` variable is used to identify each page that is tracked on the website.

If the `pageName` variable is not populated with a defined value (such as Home), Analytics uses the URL of the page. Because the page name is central to your reports, make sure that all parties in your organization agree on a strategy before you implement.

Depending on the content management system your site uses, it is helpful to add content elements to your system that can be used to populate Analytics variables. Many companies find that most Analytics variables can be populated from existing content management elements.

Page Naming Strategies

The `pageName` variable should be populated with an easy-to-read, intuitive, page identifier.

You can determine the best way of populating the `pageName` variable by looking at the structure of your website. The methods listed below outline various ways of populating the `pageName` variable.

While the `pageName` variable is central to identifying user behavior, Adobe recommends using multiple variables to indicate page information. The best page naming strategies use a different variable for each level of hierarchy within your site, as shown below:

- The `channel` variable can be used to indicate the site section.
- The `pageName` variable can be used to show content type.
- A **custom insight** variable (prop1) can be used for detailed content.

Levels of detail vary, depending on property, as shown below:

Variable	Level of Detail	Example
Channel	General Section	Electronics
Prop1	Sub Section	Sports : Local Sports
PageName	General Content Description	Loans : Home Mortgage : Rate Comparison

Variable	Level of Detail	Example
Prop2	Detailed Content Description	Electronics : Notebook PC : Detailed Specs : IBM Thinkpad T20

The more layered your site, the more variables should be used to identify page content. Companies also find value in allowing for overlap between variables. For example, a more detailed variable may contain information not only about the product being viewed, but also about the site section and sub-section. This is particularly helpful when a product or article appears in more than one section of the site.

The following page naming strategies describe how to populate the *pageName* variable. While it is tempting to choose the page naming strategy that is easiest to implement, the page naming strategy largely determines the usability of all **Path** and **Page** reports. Use good judgment when deciding how pages are named.

Unique Name for Each Page

The most valuable method of naming pages is to give each page a unique identifier that is easily understood by all Analytics users in your organization. Examples of page names include Home Page, Electronics Department Home, and Sports : Local Sports : High School.

Most Analytics users find that hierarchical page names are useful in both identifying where the page is found on the site and its purpose. The following table shows some sample page names for various industries:

Conversion	Media	Finance
Home Page	Home Page	Home Page
Electronics	Technology	Home Loans
Electronics : Notebook PCs	Technology : New Gadgets	Home Loans : Rate Compare
Electronics : Notebook PCs : Product Page	Technology : New Gadgets : Article Page	Home Loans : Rate Compare : 10 Year Fixed

File Path (Not the Full URL)

For some sites, the file path is clear and easily read. Any business user can read a URL and determine the page to which the file path refers. If this is the case for your site, you can use a server-side variable to populate the path to the file into the *pageName* variable, as shown below:

```
s.pageName="<%= file_path %>"
```

Adobe does not recommend leaving the *pageName* blank, (which results in using the full URL of the page) even though you may be tempted to do so. The following side-effects are caused by leaving the *pageName* variable blank and using the *pageURL* as the page identifier.

- The domain and path of a page may not always be displayed identically. For example, the following four URLs return a single page:

- `http://www.mysite.com/index.jsp`
- `http://www.mysite.com`
- `http://mysite.com/index.jsp`
- `http://mysite.com/`

If the *pageName* is left blank, each of these page names would occupy a separate entry in reports.

- Some pages (such as forms) post to themselves, thereby erasing any distinction between the original form and the resulting output.

- When your page is translated into another language by search engines or other online tools, the URL of the page is the URL of the search engine (not the URL of your site).

HTML (document.title)

If you have invested time into making your HTML titles readable and intuitive, you might consider using the same title as the value in the `pageName` variable. Adobe recommends using a server-side variable to populate the `pageName` rather than using JavaScript's `document.title`. Some browsers interpret the HTML title differently than others, which may cause Analytics to receive different page names from different browsers.

The best practice for using the HTML title is to copy the existing titles for each page into a separate variable or content management element. When you decide to make changes to the HTML title for search engine optimization or other purposes, the Analytics page names are not affected. If a page name changes in Analytics, it becomes a new page and is not connected with the old page name, regardless of the associated URL.

Variable Length

The length of Analytics variables can impact the size of the HTML code snippet, JavaScript library file, and image request.

If a customer has many variables that are long (60 characters or more) the values can be replaced with shorter identifiers. Data classifications or VISTA rules can be used to translate the identifiers to friendly names.



Note: Most Analytics variables have a maximum of 100 characters (eVars have a maximum of 255). Internet Explorer allows an overall maximum of 2,048 characters in a GET image request URL. The image request limit applies not only to the variables, but also to information about the browser, operating system, and browser plug-ins (Netscape/Mozilla only).

HTML Code Snippet

Many customers have variables declared, but no value is assigned to the variable.

Removing unused variables helps to reduce the page size.

Javascript Library File

The JavaScript library file is intended to be cached in the user's browser after the initial load, which limits the amount of data that needs to be downloaded.

Page-specific variables should be placed in the HTML snippet. All other variables should be put in the JavaScript library file.

Caching Directives


The JavaScript library file is intended to be cached in the user's browser after the first time the user loads a page.

Adobe customers should ensure that their Web servers are set up to take advantage of this functionality. For example, make sure that the NO-CACHE setting is set to false. Additionally, ensure that the expiration date is sufficiently long. Make sure any proxy caches are set up with the correct configuration. The customer's Web server documentation provides more information.

Tables

Many Web browsers do not start displaying the contents of a table until the browser has compiled the entire table. If the entire contents of the page are inside one big table, the browser must compile the entire contents of the page before anything is displayed.


Placing the call to the JavaScript library file outside table tags ensures that the call to the Analytics servers does not impact the displaying of the page content.

 **Note:** *The file should be placed in a legal position for images and must appear between the opening <body> tag and the closing </body> tag.*

File Compression

Customers can compress the JavaScript library file by using standards-based encoding (such as gzip).

Typical compression algorithms can reduce the size of the JavaScript file by 40-60% or more.

 **Note:** *Not all browsers support all file compression standards or interpret the compressed files in the same manner. Adobe does not guarantee reliable file compression in all environments. Customers should test compression in their supported browsers and configurations before deploying.*

Secure Pages

Secure pages (pages loaded under https://) encrypt the image request and add to the total download time.

Secure pages can add as much as 50-75% overhead to the image request.

Content Delivery Services/Networks

Content Delivery Services or Content Distribution Networks (CDNs) such as Akamai and Speedera push Web content closer to the edge of the network, keeping frequently-requested documents close to the location where they are accessed.

Typically, this reduces access latency, bandwidth usage, and infrastructure cost.

Your AppMeasurement for JavaScript library file can be delivered via a CDN to enhance performance and delivery of the file to the site visitor. Adobe customers need to ensure they have configured their CDN services correctly. CDNs are a common reason for fluctuations in download times and should be considered the most probable cause for any changes in download times.

Tag manager stores all JavaScript on a CDN.

JavaScript File Location and Concurrency

Placing the call to the JavaScript library file at the top of the page ensures that the image is among the first elements to be downloaded.

Most Web browsers download images concurrently. Typically three to four images can be downloaded simultaneously.

Because most Web browsers download elements concurrently, the Status Bar of many common browsers (including Internet Explorer) does not accurately reflect which element the browser is trying to load. For example, your Status

Bar may report that your browser is waiting for image 1 to download. The network packet tests show your browser has already received image 1 and is currently waiting for image 2.



Note: Because third-party Internet Performance Audit providers (such as Keynote Systems) download page image elements sequentially, not concurrently, they do not mimic the typical user experience.

Peering

Private Network Peering enables data to pass from an ISP's network to the Analytics network more efficiently.

Peering is simply the agreement to interconnect and exchange routing information without the need of the public network. Please contact Adobe Engineering if an ISP is interested in establishing a peering relationship.

Although peering may provide some benefits to Adobe ISP customers, optimizing the HTML snippet and the Web servers caching directives likely have a much greater impact. High-volume ISP customers realize the most benefit from peering.

Report to Variable Mapping

The tables below display the report to variable mapping, or the reports and the variables that are used in them.

Conversion Reports

The following table lists the conversion variables that are used to populate each report:

Purchases		
Conversions and Averages	s.events, s.products, s.purchaseID	Set s.events to purchase, product detail, order number
Revenue	s.events, s.products, s.purchaseID	Set s.events to purchase, product detail, order number
Orders	s.events, s.products, s.purchaseID	Set s.events to purchase, product detail, order number
Units	s.events, s.products, s.purchaseID	Set s.events to purchase, product detail, order number

Shopping Cart		
Conversions and Averages	s.events, s.products, s.purchaseID	
Cart	s.events	Set s.events to scOpen
Cart Views	s.events	Set s.events to scView
Cart Additions	s.events	Set s.events to scAdd
Cart Removals	s.events	Set s.events to scRemove
Checkouts	s.events	Set s.events to scCheckout

Custom Events		
Custom Event 1	s.events	Populate with event1 - event100
...	...	Populate with event1 - event100

Custom Events		
Custom Event 100	s.events	Populate with event1 - event100

Products		
Conversions and Averages	s.events, s.products, s.purchaseID	
Products	s.products, s.events	May vary depending on right column metrics
Cross Sell	s.products, s.events, s.purchaseID	May vary depending on right column metrics
Categories	s.products	May vary depending on right column metrics

Campaigns		
Conversions and Averages	s.products, s.events, s.campaign	
Tracking Code	s.campaign	
Creative Elements	N/A	Defined in Analytics
Campaigns	N/A	Defined in Analytics

Customer Loyalty		
Customer Loyalty	s.products, s.events, s.purchaseID	Variables set on the Order Confirmation (Thank You!) page

Sales Cycle		
Days Before First Purchase	s.products, s.events, s.purchaseID	Variables set on the Order Confirmation (Thank You!) page
Days Since Last Purchase	s.products, s.events, s.purchaseID	Variables set on the Order Confirmation (Thank You!) page
Visit Number	s.products, s.events, s.purchaseID	Variables set on the Order Confirmation (Thank You!) page
Daily Unique Customers	s.products, s.events, s.purchaseID	Variables set on the Order Confirmation (Thank You!) page
Monthly Unique Customers	s.products, s.events, s.purchaseID	Variables set on the Order Confirmation (Thank You!) page
Yearly Unique Customers	s.products, s.events, s.purchaseID	Variables set on the Order Confirmation (Thank You!) page

Finding Methods		
Referring Domains	N/A	Automatically set by the .JS file
Original Referring Domains	N/A	Automatically set by the .JS file
Search Engines	N/A	Automatically set by the .JS file
Search Keywords	N/A	Automatically set by the .JS file

Visitor Profile		
Top Level Domains	N/A	Automatically set by the .JS file
Languages	N/A	Automatically set by the .JS file
Time Zones	N/A	Automatically set by the .JS file
States	s.state	Variable set on the Order Confirmation (Thank You!) page
Zip/Postal Codes	s.zip	Variable set on the Order Confirmation (Thank You!) page
Domains	N/A	Automatically set by the .JS file

Technology		
Browsers	N/A	Automatically set by the .JS file
Operating Systems	N/A	Automatically set by the .JS file
Monitor Resolutions	N/A	Automatically set by the .JS file

Site Path		
Page Value	s.pageName, s.products, s.events, s.purchaseID	
Entry Pages	s.pageName	
Original Entry Pages	s.pageName	
Pages Per Visit	N/A	Calculated by business rules in Analytics
Time Spent on Site	N/A	Calculated by business rules in Analytics
Site Sections	s.channel	Same as Channel report in Traffic reports section

Customer Variables		
Custom eVar 1	s.eVar 1	
Custom eVar 2	s.eVar 2	
Custom eVar 3	s.eVar 3	
...		
Custom eVar 75	s.eVar 75	

Traffic Reports

The following table lists the **traffic** variables that are used to populate each report:

Calculated Metrics		
N/A	N/A	N/A

Site Traffic		
Page Views	N/A	Calculated by business rules in Analytics
Hourly Unique Visitors	N/A	Calculated by business rules in Analytics
Daily Unique Visitors	N/A	Calculated by business rules in Analytics
Monthly Unique Visitors	N/A	Calculated by business rules in Analytics
Yearly Unique Visitors	N/A	Calculated by business rules in Analytics
Visits	N/A	Calculated by business rules in Analytics
File Downloads	N/A	Automatically tracked by .JS file (depends on .JS variable settings)

Finding Methods		
Referring Domains	N/A	Automatically set by the .JS file
Referrers	N/A	Automatically set by the .JS file
Search Engines	N/A	Automatically set by the .JS file
Search Keywords	N/A	Automatically set by the .JS file
Return Frequency	N/A	Calculated by business rules in Analytics
Daily Return Visits	N/A	Calculated by business rules in Analytics
Return Visits	N/A	Calculated by business rules in Analytics
Visit Numbers	N/A	Calculated by business rules in Analytics

Visitor Profile		
Domains	N/A	Automatically set by the .JS file
Top Level Domains	N/A	Automatically set by the .JS file
Languages	N/A	Automatically set by the .JS file
Time Zones	N/A	Automatically set by the .JS file
Visitor Details	N/A	Automatically set by the .JS file
Last 100 Visitors	N/A	Calculated by business rules in Analytics
User Home Page	N/A	Automatically set by the .JS file
Key Visitors	N/A	Based on visitor's IP address

Visitor Profile		
Pages Viewed by Key Visitors	N/A	Based on visitor's IP address

Geo Segmentation		
Countries	N/A	Based on visitor's IP address
U.S. States	N/A	Based on visitor's IP address
DMA	N/A	Based on visitor's IP address
International Cities	N/A	Based on visitor's IP address
U.S. Cities	N/A	Based on visitor's IP address

Technology		
Browser Types	N/A	Automatically set by the .JS file
Browsers	N/A	Automatically set by the .JS file
Mobile Devices	N/A	Automatically set by the .JS file
Browser Width	N/A	Automatically set by the .JS file
Browser Height	N/A	Automatically set by the .JS file
Operating Systems	N/A	Automatically set by the .JS file
Monitor Color Depth	N/A	Automatically set by the .JS file
Monitor Resolutions	N/A	Automatically set by the .JS file
Netscape Plug-ins	N/A	Automatically set by the .JS file
Java	N/A	Automatically set by the .JS file
JavaScript	N/A	Automatically set by the .JS file
JavaScript Version	N/A	Automatically set by the .JS file
Cookies	N/A	Automatically set by the .JS file
Connection Types	N/A	Automatically set by the .JS file
Segmentation		

Segmentation		
Most Popular Pages	s.pageName	
Most Popular Site Sections	s.channel	
Most Popular Servers	s.server	

Custom Insight		
Custom Links	s.linkName	Requires custom implementation
Custom Insight 1	s.prop1	
...	...	
Custom Insight 75	s.prop75	

Hierarchies		
Hierarchy 1	s.hier1	
...	...	
Hierarchy 5	s.hier5	

Pathing Reports

The following table lists the pathing variables that are used to populate each report within Analytics:

Pages		
Page Summary	s.pageName (or other pathed variable)	Also depends on internal business rules
Page Value	s.pageName (or other pathed variable)	Also depends on internal business rules
Most Popular Pages	s.pageName (or other pathed variable)	Also depends on internal business rules
Reloads	s.pageName (or other pathed variable)	Also depends on internal business rules
Clicks to Page	s.pageName (or other pathed variable)	Also depends on internal business rules
Time Spent on Page	s.pageName (or other pathed variable)	Also depends on internal business rules
Pages Not Found	s.pageName (or other pathed variable)	Also depends on internal business rules

Entries and Exits		
Entry Pages	s.pageName (or other pathed variable)	Also depends on internal business rules
Exit Pages	s.pageName (or other pathed variable)	Also depends on internal business rules
Exit Links	s.pageName (or other pathed variable)	Also depends on internal business rules

Complete Paths		
Full Paths	s.pageName (or other pathed variable)	Also depends on internal business rules
Longest Paths	s.pageName (or other pathed variable)	Also depends on internal business rules
Path Length	s.pageName (or other pathed variable)	Also depends on internal business rules
Time Spent per Visit	s.pageName (or other pathed variable)	Also depends on internal business rules
Single-page Visits	s.pageName (or other pathed variable)	Also depends on internal business rules

Advanced Analysis		
Previous Page	s.pageName (or other pathed variable)	Also depends on internal business rules
Next Page	s.pageName (or other pathed variable)	Also depends on internal business rules
Previous Page Flow	s.pageName (or other pathed variable)	Also depends on internal business rules
Next Page Flow	s.pageName (or other pathed variable)	Also depends on internal business rules
PathFinder	s.pageName (or other pathed variable)	Also depends on internal business rules
Fall-out	s.pageName (or other pathed variable)	Also depends on internal business rules

Variable to Report Mapping

The table below displays variable to report mapping for the variables used to populate Analytics reports.

Each variable is listed with the reports that use the variable listed next to it.

Variable	Reports Populated
s.pageName	<ul style="list-style-type: none"> Conversion Reports > Path Reports > Page Value Conversion Reports > Path Reports > Entry Page Conversion Reports > Path Reports > Original Entry Pages Traffic Reports > Site Traffic > Page Views Traffic Reports > Site Traffic > Hourly Unique Visitors Traffic Reports > Site Traffic > Daily Unique Visitors Traffic Reports > Site Traffic > Monthly Unique Visitors Traffic Reports > Site Traffic > Yearly Unique Visitors Traffic Reports > Visitor Profile > Pages Viewed by Key Visitors Traffic Reports > Segmentation > Most Popular Pages Path Reports > Pages > Page Summary Path Reports > Pages > Page Value Path Reports > Pages > Most Popular Pages Path Reports > Pages > Reloads Path Reports > Pages > Click to Page Path Reports > Pages > Time Spent on Page Path Reports > Entries & Exits > Entry Pages

Variable	Reports Populated
	Path Reports > Entries & Exits > Exit Pages Path Reports > Entries & Exits > Exit Links Path Reports > Complete Paths > Full Paths Path Reports > Complete Paths > Longest Paths Path Reports > Complete Paths > Path Length Path Reports > Complete Paths > Time Spent per Visit Path Reports > Complete Paths > Single-page Visits Path Reports > Advanced Analysis > Previous Page Path Reports > Advanced Analysis > Next Page Path Reports > Advanced Analysis > Previous Page Flow Path Reports > Advanced Analysis > Next Page Flow Path Reports > Advanced Analysis > PathFinder Path Reports > Advanced Analysis > Fall-out NOTE: In all of the Traffic Reports listed above, with the exception of the Most Popular Pages Report, if the s.pageName variable is not set, the URL is used.
s.server	Traffic Reports > Segmentation > Most Popular Servers
s.pageType	Path Reports > Pages > Pages Not Found
s.channel	Conversion Reports > Site Path Reports > Site Section Traffic Reports > Segmentation > Most Popular Site Sections
s.prop1 - s.prop20	Traffic Reports > Custom Insight > Custom Insight 1-20
s.campaign	Conversion Reports > Campaigns > Conversion & Averages Conversion Reports > Campaigns > Tracking Code
s.state	Conversion Reports > Visitor Profile > States
s.zip	Conversion Reports > Visitor Profile > ZIP/ Postal Codes
s.events	Conversion Reports > Purchases > Conversion & Averages Conversion Reports > Purchases > Revenue Conversion Reports > Purchases > Orders Conversion Reports > Purchases > Units Conversion Reports > Shopping Cart > Conversion & Averages Conversion Reports > Shopping Cart > Carts

Variable	Reports Populated
	<p>Conversion Reports > Shopping Cart > Cart Views</p> <p>Conversion Reports > Shopping Cart > Cart Additions</p> <p>Conversion Reports > Shopping Cart > Cart Removals</p> <p>Conversion Reports > Shopping Cart > Checkouts</p> <p>Conversion Reports > Custom Events > Custom Event 1-20</p> <p>Conversion Reports > Products > Conversion & Averages</p> <p>Conversion Reports > Products > Cross Sell</p> <p>Conversion Reports > Products > Categories</p> <p>Conversion Reports > Campaigns > Conversion & Averages</p> <p>Conversion Reports > Customer Loyalty > Customer Loyalty</p> <p>Conversion Reports > Sales Cycle > Days Before First Purchase</p> <p>Conversion Reports > Sales Cycle > Days Since Last Purchase</p> <p>Conversion Reports > Sales Cycle > Visit Number</p> <p>Conversion Reports > Sales Cycle > Daily Unique Customers</p> <p>Conversion Reports > Sales Cycle > Monthly Unique Customers</p> <p>Conversion Reports > Sales Cycle > Yearly Unique Customers</p> <p>Conversion Reports > Site Path > Page Value</p>
s.products	<p>Conversion Reports > Purchases > Conversion & Averages</p> <p>Conversion Reports > Purchases > Revenue</p> <p>Conversion Reports > Purchases > Orders</p> <p>Conversion Reports > Purchases > Units</p> <p>Conversion Reports > Shopping Cart > Conversion & Averages</p> <p>Conversion Reports > Products > Conversion & Averages</p> <p>Conversion Reports > Products > Cross Sell</p> <p>Conversion Reports > Products > Categories</p> <p>Conversion Reports > Campaigns > Conversion & Averages</p> <p>Conversion Reports > Customer Loyalty > Customer Loyalty</p> <p>Conversion Reports > Sales Cycle > Days Before First Purchase</p> <p>Conversion Reports > Sales Cycle > Days Since Last Purchase</p> <p>Conversion Reports > Sales Cycle > Visit Number</p> <p>Conversion Reports > Sales Cycle > Daily Unique Customers</p>

Variable	Reports Populated
	Conversion Reports > Sales Cycle > Monthly Unique Customers Conversion Reports > Sales Cycle > Yearly Unique Customers Conversion Reports > Site Path > Page Value
s.purchaseID	Conversion Reports > Purchases > Conversion & Averages Conversion Reports > Purchases > Revenue Conversion Reports > Purchases > Orders Conversion Reports > Purchases > Units Conversion Reports > Shopping Cart > Conversion & Averages Conversion Reports > Products > Conversion & Averages Conversion Reports > Products > Cross Sell Conversion Reports > Customer Loyalty > Customer Loyalty Conversion Reports > Sales Cycle > Days Before First Purchase Conversion Reports > Sales Cycle > Days Since Last Purchase Conversion Reports > Sales Cycle > Visit Number Conversion Reports > Sales Cycle > Daily Unique Customers Conversion Reports > Sales Cycle > Monthly Unique Customers Conversion Reports > Sales Cycle > Yearly Unique Customers Conversion Reports > Site Path > Page Value
s.eVar1 - s.eVar20	Conversion Reports > Custom Variables > Customer eVar 1-20
s.linkName	Traffic Reports > Custom Insight > Custom Links
s.hier1 - s.hier5	Traffic Reports > Hierarchies > Hierarchy 1-5

Analytics for Digital Assistants

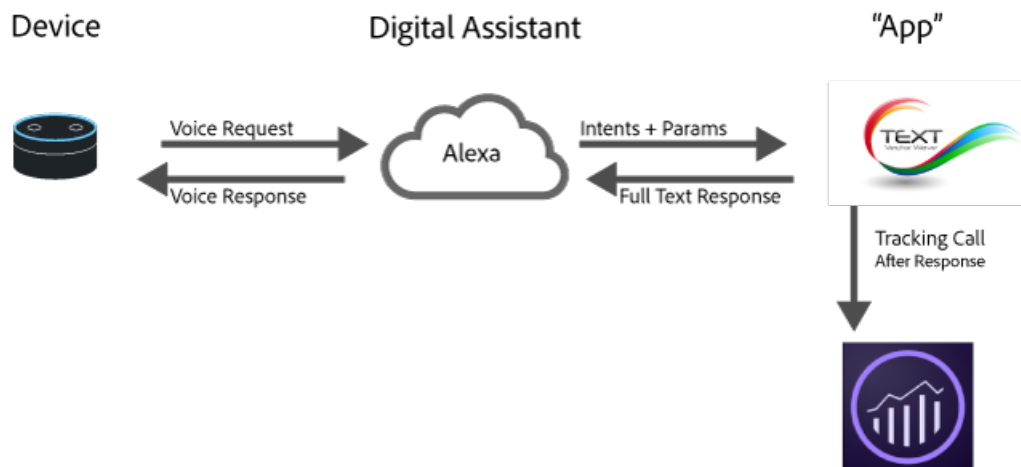
With recent advances in cloud computing, machine learning, and natural language processing, digital assistants are moving out of the dark ages of "clippy" and becoming part of everyday life. Consumers are starting to talk to their devices and expecting them to listen, understand, and respond in natural, human-like ways to phrases like, "Alexa, turn on the family room lights," or "Okay Google, What's the weather like outside?"

As these platforms become more established, brands can present their services to consumers in these same realistic and lifelike ways. For example, consumers can ask things like:

- "Alexa, ask my car when it needs an oil change."
- "Cortana, what is the balance of my checking account?"
- "Siri, send John \$20 for dinner last night from my banking app."

This white paper provides an overview of how best to use the Adobe Analytics Cloud to measure and optimize these types of experiences.

Digital Experience Architecture Overview



Most digital assistants today follow a similar high-level architecture:

1. **Device:** There is a device (like an Amazon Echo or a phone) with a microphone that allows the user to ask a question.
2. **Digital assistant:** That device interacts with the service that powers the digital assistant. This service is where a lot of the magic happens. It is where the speech is converted into machine understandable intents and the details of the request are parsed out. Once the user's intent is understood, the digital assistant passes the intent and details of the request to the app that handles the request.
3. **"App":** The app can either be an app on the phone or a voice app. The app is responsible for responding to the request. It responds to the digital assistant and the digital assistant then responds to the user.

Where to implement Analytics

One of the best places to implement Analytics is in the app. The app receives the *intent* and the details about the intent from the digital assistant and decides how to respond.

There are two times during the life-cycle of a request that can be helpful to call the Analytics Cloud.

1. When the request is sent to the "App." If you need additional context about the user before you respond to the request, you should leverage the Audience Manager capability to get the segments that they belong to.
2. After the response is returned from the app. If you are just interested in recording what happened with the customer for future optimization, send a request to Adobe Analytics after the response has been returned. This way you have the full context of what the request was and how the system responded.

Analytics Implementation for Digital Assistants

New Installs

For some of the digital assistants you will get a notification when someone installs the skill. This is especially the case when there is authentication involved. At this time you should send Adobe an *Install* event by setting the context data to `a.InstallEvent=1`. Note that this is not available on all platforms but is helpful when it is present for looking at retention. The following code sample sends in *Install*, *Install Date*, and *AppID*.

Code Sample

```
GET
/b/ss/[rsid]/0?vid=[UserID]&c.a.InstallEvent=1&c.a.InstallDate=2017-04-24&c.a.AppID=Spoofify1.0&c.a.OSType=Alexa&pageName=install
HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

Multiple Assistants or Multiple Apps

It is likely that you will develop apps for multiple platforms. The best practice is to include an app ID with each request. This can be set in the `a.AppID` context data. Follow the format of `[AppName] [BundleVersion]`, for example, `BigMac for Alexa 1.2`

Code Sample

```
GET
/b/ss/[rsid]/0?vid=[UserID]&c.a.AppID=Spoofify1.0&c.a.Launches=1&c.Product=AmazonEcho&c.OSType=Alexa&pageName=install
HTTP/1.1
Host: [prefix].sc.omtrdc.net
Cache-Control: no-cache
```

```
GET
/b/ss/[rsid]/0?vid=[UserID]&c.a.AppID=Spoofify2.0&c.a.Launches=1&c.Product=GoogleHome&c.OSType=Android&pageName=install
HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

User/Visitor Identification

The Analytics Cloud uses the [Experience Cloud ID service](#) (ECID) service to tie interactions across time to the same person. Most of the digital assistants will return a *userID* that you can use to keep the activity for different users separate in the ECID service. In most cases, this is what you should pass in as the ECID service. Some platforms return a *userID* that is longer than the 100 characters that the Analytics limit allows. If that is the case, Adobe recommends that you hash the *userID* to a fixed-length value using a standard hashing algorithm, such as MD5 or Sha1.

While you can use the ECID service for this, doing so provides value only if you are trying to map ECIDs across different devices (e.g. Web to Digital Assistant). If your app is a mobile app (e.g. a deep link) you should use the SDK as is and send the information along. The *userID* can be attached to the ECID service using the `setCustomerID` method, allowing for better device stitching. However, if your app is a service, use the *userID* provided by the service as the ECID, as well as setting it in the `setCustomerID`. This allows you to see how people are using the digital assistant over time.

Code Sample

```
GET /b/ss/[rsid]/0?vid=[UserID]&pageName=[intent] HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

Sessions

Because digital assistants are conversational, they often have the concept of a session. For example:

Consumer: "Ok Google, call a cab for me"

Google:: "Sure, what time would you like?"

Consumer: "8:30pm"

Google: "Sounds good, the Driver will be by at 8:30pm"

These sessions are important to keep in context. They help collect more details and make the digital assistants more natural.

When implementing Analytics on a conversation, there are two things you should do when a new session is started:

1. **Reach out to Audience Manager:** Get the relevant segments that a user is a part of so that you can customize the response. (For example, this person currently qualifies for the multi-channel discount.)
2. **Send in a new session or launch event:** When you send the first response to Analytics, include a launch event. Usually, this can be sent by setting context data of `a.LaunchEvent=1`.

Code Sample for Launch

```
GET /b/ss/[rsid]/0?vid=[UserID]&c.a.LaunchEvent=1&c.Intent=[intent]&pageName=[intent] HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

Intents

Each of the digital assistants has algorithms that detect intents and then passes the intent down to the "App" so that the app knows what to do. These intents are a succinct representation of the request.

For example, if a user says, "Siri, Send John \$20 for dinner last night from my banking app," the intent might be something like *sendMoney*.

By sending in each of these requests as an eVar, you will be able to run pathing reports on each of the intents for conversational apps. You will also want to make sure the "App" can handle requests without an intent. We recommend passing in *No Intent Specified* as opposed to leaving the value blank.

Code Sample

```
GET
/b/ss/[rsid]/0?vid=[UserID]&c.a.AppID=Penmol.0&c.a.LaunchEvent=1&c.Intent=SendPayment&pageName=[intent]
HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

or

```
GET
/b/ss/[rsid]/0?vid=[UserID]&c.a.AppID=Penmol.0&c.a.LaunchEvent=1&c.Intent=No_Intent_Specified&pageName=[intent]
HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

Parameters/Slots/Entities

In addition to the intent the digital assistant will often have a set of key value pairs that give the details of the intent. These are called slots, entities or parameters. For example:

"Siri, Send John \$20 for dinner last night from my banking app" would have the following parameters:

- Who = John
- Amount = 20
- Why = Dinner

There is usually a finite number of these with your app. To track these in Analytics, send them into context data and then map each of the parameters to an eVar.

Code Sample

```
GET
/b/ss/[rsid]/0?vid=[UserID]&c.a.AppID=Penmol.0&c.a.LaunchEvent=1&c.Intent=SendPayment&c.Amount=20.00&c.Reason=Dinner&c.ReceivingPerson=John&c.Intent=SendPayment&pageName=[intent]
HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

Session

Although not all apps will be revenue generating, it is important to think about what success looks like and include some measurements for it. Adobe Analytics can measure revenue, ad-impressions and other forms of success along with the user behavior.

Error States

Sometimes the Digital Assistant will provide the app with inputs that it doesn't know how to handle. For example.

"Siri, Send John 20 bags of coal for dinner last night from my banking app"

When this happens the app should ask for clarification. Additionally when it is responding to a request like this you should send Analytics an event that indicates the App has an error state along with an evar that specifies what type of error occurred.

Be sure to include errors where the inputs are not correct and errors where the "App" had a problem.

Code Sample

```
GET
/b/ss/[rsid]/0/?vid=[UserID]&c.a.AppID=Penm01.0&c.Error=1&c.ErrorName=InvalidCurrency&pageName=[intent]
HTTP/1.1
Host: sc.omtrdc.net
Cache-Control: no-cache
```

Device Capabilities

While most of the platforms don't expose the actual device that the user spoke too. They do expose the capabilities of the device as the available interfaces (e.g. Audio, Screen, Video, etc). This is useful information because it defines the types of content that can be used when interacting with your users. When measuring the interfaces it is best to concatenate them (in alphabetical order).

Example: :Audio:Camera:Screen:Video:

Notice the trailing and leading colons. These help when creating segments (for example, give me all interactions with :Audio: capabilities).

Amazon Capabilities:

<https://developer.amazon.com/public/solutions/alexa/alexa-skills-kit/docs/alexa-skills-kit-interface-reference>

Google Capabilities: <https://developers.google.com/actions/assistant/surface-capabilities>

Analytics Reporting for Digital Assistants

Once your Digital Assistant App is implemented you can use the full power of Adobe Analytics with it. Below are just a few examples of the things you can do with Analytics.

Monitoring Intents

Most Apps have several intents and different things you can do. You could easily use Analysis Workspace to keep track of the top intents by instances and by users

This lets you see which features are being used most often and can give you a view into the adoption of new features.

Requests with Errors

You will be able to monitor errors to see if there are common places where users are getting having problems.

Flow Between Events

One of the most powerful things to do is look at the flow of intents. This is helpful in two ways. First, you can look within a session for how people flow between intents in a conversation. Second, you can look at how people flow between intents over longer time frames to see how their usage of the "App" is evolving.

Example

Pre-installed app

Person	Device Response	Action / Intent	Get Request	Analytics Data
Play Spoofify	"okay playing Spoofify"	Play	GET https://sc.omtrdc.net/spotify/turnMusicOn HTTP/1.1 Host: sc.omtrdc.net Cache-Control: no-cache	<ul style="list-style-type: none"> • Visitor ID • App Version • # of Launches • Intent • Response
Change song	"okay what song do you want?"	ChangeSong	GET https://sc.omtrdc.net/spotify/turnMusicOn AskForSong HTTP/1.1 Host: sc.omtrdc.net Cache-Control: no-cache	<ul style="list-style-type: none"> • Visitor ID • App Version • Intent • *blank playlist • Response
Play "My Heart Will Go On" by Celine Dion	"okay playing 'My Heart Will Go On' by Celine Dion"	ChangeSong	GET https://sc.omtrdc.net/spotify/turnMusicOn ActionPlaySong&c.SongID=[012345] HTTP/1.1 Host: sc.omtrdc.net Cache-Control: no-cache	<ul style="list-style-type: none"> • Visitor ID • App Version • Intent • Song ID • Response
Change playlist	"okay what playlist do you want?"	ChangePlaylist	GET https://sc.omtrdc.net/spotify/turnMusicOn AskForPlaylist HTTP/1.1 Host: sc.omtrdc.net Cache-Control: no-cache	<ul style="list-style-type: none"> • Visitor ID • App Version • Intent • *blank playlist • Response
Play Usher	"okay playing Usher"	ChangePlaylist	GET https://sc.omtrdc.net/spotify/turnMusicOn ActionPlayPlaylist&c.Playlist=Usher HTTP/1.1 Host: sc.omtrdc.net Cache-Control: no-cache	<ul style="list-style-type: none"> • Visitor ID • App Version • Intent • Playlist • Response
Turn music off	*no response, music turns off	Off	GET https://sc.omtrdc.net/spotify/turnMusicOn HTTP/1.1 Host: sc.omtrdc.net Cache-Control: no-cache	<ul style="list-style-type: none"> • Visitor ID • App Version • Intent • Response

Requires user to install app

Contact and Legal Information

Information to help you contact Adobe and to understand the legal issues concerning your use of this product and documentation.

Help & Technical Support

The Adobe Experience Cloud Customer Care team is here to assist you and provides a number of mechanisms by which they can be engaged:

- [Check the Experience Cloud help pages for advice, tips, and FAQs](#)
- [Ask us a quick question on Twitter @AdobeExpCare](#)
- [Log an incident in our customer portal](#)
- [Contact the Customer Care team directly](#)
- [Check availability and status of Experience Cloud Solutions](#)

Service, Capability & Billing

Dependent on your solution configuration, some options described in this documentation might not be available to you. As each account is unique, please refer to your contract for pricing, due dates, terms, and conditions. If you would like to add to or otherwise change your service level, or if you have questions regarding your current service, please contact your Account Manager.

Feedback

We welcome any suggestions or feedback regarding this solution. Enhancement ideas and suggestions [can be added to our Customer Idea Exchange](#).

Legal

© 2018 Adobe Systems Incorporated. All Rights Reserved.
Published by Adobe Systems Incorporated.

[Terms of Use](#) | [Privacy Center](#)

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. A trademark symbol (®, ™, etc.) denotes an Adobe trademark.

All third-party trademarks are the property of their respective owners. Updated Information/Additional Third Party Code Information available at <http://www.adobe.com/go/thirdparty>.