



Adobe® Marketing Cloud SiteCatalyst Implementation

Contents

SiteCatalyst Implementation.....	12
Moving Data in and out of SiteCatalyst.....	14
Tracking Online Data.....	18
Tracking Web Pages.....	18
Tracking Links and Page Elements.....	19
Tracking Rich Media - The t() and The tl() Functions.....	20
ActionSource.....	21
Media Tracking.....	22
Tracking Links as Web Pages.....	22
Using the Data Insertion API.....	22
Tracking Offline Data.....	23
JavaScript Implementation.....	25
How does SiteCatalyst Collect and Report Data?.....	25
Step 1 - Create a New Report Suite.....	25
Step 2 - Generate Code.....	26
Step 3 - Modify Your Web Page.....	26
Step 4 - Validate Your Implementation.....	28
Tracking Across Different Implementation Types.....	29
Implementation Guidelines.....	29
Implementation Example.....	30
Collecting Data From Form Elements.....	31
AJAX-Tracking Rich Media Applications.....	32
Implementing with AJAX.....	33
Implementing SiteCatalyst without JavaScript.....	36

PHP Measurement Library.....	36
Variable Names.....	37
Other Requirements.....	37
Sample Code.....	39
Supported SiteCatalyst Reports.....	40
Implementing SiteCatalyst on Mobile Sites.....	42
Video Measurement.....	44
Processing Rules.....	45
Other AppMeasurement Libraries.....	46
TagManager.....	47
How TagManager Works.....	47
Tag Container Environments.....	48
Asynchronous Loading.....	50
TagManager Quick Start.....	50
Migrating to a Tag Container.....	52
SiteCatalyst.....	53
Marketing Cloud Deployment.....	54
Survey.....	54
AudienceManager	54
JavaScript Video Measurement.....	54
Managing Tags.....	54
Add a Genesis Integration.....	55
Add a Custom Integration.....	55
TagManager Reference.....	55
Custom Core JavaScript.....	56
Product Code.....	56
DoPlugins.....	56
Dependent Code JavaScript.....	56
Custom Code (after products).....	56

Processing Order.....	57
Conditional Execution.....	58
Permissions.....	59

Additional Implementation Resources.....60

Vulnerability Scanner.....61

SiteCatalyst Variables.....62

Variables and Limitations.....	62
Illegal JavaScript Characters.....	66
s_account.....	66
browserHeight.....	67
browserWidth.....	68
campaign.....	68
charSet.....	69
channel.....	70
colorDepth.....	70
connectionType.....	71
cookieDomainPeriods.....	71
cookieLifetime.....	72
cookiesEnabled.....	73
currencyCode.....	73
dc.....	74
doPlugins.....	74
dynamicAccountList.....	75
dynamicAccountMatch.....	76
dynamicAccountSelection.....	77
dynamicVariablePrefix.....	78
eVarN.....	78
events.....	81
fpCookieDomainPeriods.....	83

hierN.....	84
homepage.....	85
javaEnabled.....	85
javascriptVersion.....	85
linkDownloadFileTypes.....	86
linkExternalFilters.....	87
linkInternalFilters.....	88
linkLeaveQueryString.....	89
linkName.....	90
linkTrackEvents.....	90
linkTrackVars.....	91
linkType.....	92
List Props.....	93
List Variable.....	94
maxDelay.....	95
mediaLength.....	96
mediaName.....	97
mediaPlayer.....	98
mediaSession.....	98
Media.trackEvents.....	99
Media.trackVars.....	100
mobile.....	100
pageName.....	100
pageType.....	101
pageURL.....	102
plugins.....	102
products.....	103
propN.....	105
purchasID.....	106
referrer.....	106
resolution.....	107

s_objectID.....	108
server.....	109
state.....	109
timestamp.....	110
trackDownLoadLinks.....	111
trackExternalLinks.....	112
trackingServer.....	112
trackingServerSecure.....	113
trackInlineStats.....	113
transactionID.....	113
s_usePlugins.....	114
visitorID.....	115
visitorNamespace.....	115
zip.....	116

Context Data Variables.....117

Variable Overrides.....119

Plug-ins.....120

appendList.....	120
Cookie Combining Utility.....	122
detectRIA.....	122
doPlugins Function.....	124
getAndPersistValue.....	125
getDaysSinceLastVisit.....	125
getNewRepeat.....	127
getPercentPageViewed.....	128
getPreviousValue.....	129
getQueryParam.....	130
getTimeParting.....	132
getValOnce.....	134

getVisitNum.....	135
manageQueryParam.....	136
trackTNT.....	136
The s.t() Function.....	137
The s.sa() Function.....	138
Link Tracking.....	139
Automatic Tracking of Exit Links and File Downloads.....	139
Manual Link Tracking Using Custom Link Code.....	140
Link Tracking Using an Image Request.....	143
Setting Additional Variables for File Downloads, Exit Links, and Custom Links.....	144
Using Function Calls with Custom Link Code.....	145
Validating File Downloads, Exit Links, and Custom Links.....	145
Tracking Links with First-Party Cookies.....	146
Traffic props and Conversion eVars.....	147
Using props vs. eVars.....	147
Using props as Counters.....	147
Counting Content Hierarchies.....	148
What is a Predefined Event?.....	149
Detailed Product View Page.....	149
What is a Custom Event?.....	150
Email Campaign Tracking.....	150
Implementing Email Campaign Tracking.....	151
Tracking External Email.....	152
Products.....	152
When Should the Product Variable be Set?.....	153
Setting a Product with an Event.....	153
Setting a Product without an Event.....	153
Field Definitions of the Product String.....	153
Sample Product Strings.....	154

Pathing.....	155
Enabling Pathing on a prop.....	155
Pathing by Campaign or Tracking Code.....	155
Reasons Pathing may not be Recorded.....	156
Moving from Section to Section.....	156
Moving from Page Template to Page Template.....	156
Segmenting Paths by User Type.....	157
Using Implementation Plug-ins.....	158
Calling Plug-ins with doPlugins Function.....	158
Implementation Plug-ins.....	159
DigitalPulse Debugger.....	161
Automatic Setup.....	161
Manual Setup.....	161
Mozilla Firefox.....	161
Google Chrome.....	162
Microsoft Internet Explorer.....	162
Launching DigitalPulse Debugger.....	162
Dynamic Variables and Query String Parameters.....	163
Packet Analyzers.....	167
Testing and Validation.....	168
Identifying the s_account Variable in the DigitalPulse Debugger.....	168
Deployment Validation.....	168
JavaScript JS File.....	169
Code Modifications.....	169
Variables and Values.....	170
Custom Variables.....	170
Implementation Acceptance.....	171
Data Accuracy Validation.....	172

Common Syntax Mistakes in SiteCatalyst.....174

Putting SiteCatalyst Code in the Head Tag.....	174
Using s.linkTrackVars and s.linkTrackEvents.....	175
Common Mistakes in the Products Variable.....	176
Setting the PageType Variable Correctly.....	177
Using White Space in Variable Values.....	177
Using Quotes.....	178
Replacing Your SiteCatalyst Code.....	179
Table of Common Syntax Mistakes.....	179

Optimizing your SiteCatalyst Implementation.....181

Variable Length.....	181
HTML Code Snippet.....	181
Javascript Library File.....	181
Caching Directives.....	181
Tables.....	182
File Compression.....	182
Secure Pages.....	182
Content Delivery Services/Networks.....	182
JavaScript File Location and Concurrency.....	182
Peering.....	183

Page Naming.....184

Page Naming Strategies.....	184
Optimum Path Engine.....	185
Changing Page Names in SiteCatalyst.....	186
Unique Value Limits.....	187

Dynamic Variables.....188

Dynamic Variable Examples.....	189
Other Uses for Dynamic Variables.....	189

Dynamic Variable Abbreviations.....	189
Identifying Unique Visitors.....	192
Cross-Device Visitor Identification.....	194
Connecting Users Across Devices.....	195
Data Impact of Cross-Device Visitor Identification.....	195
Example Visit.....	195
Visitors.....	196
Visits.....	197
Segments.....	197
Geo-Segmentation Data.....	197
Persisted Dimensions (eVars, Marketing Channels, Campaign).....	197
Redirects and Aliases.....	198
SiteCatalyst and Redirects.....	198
Implementing Redirects for SiteCatalyst.....	200
Configure Referrer Override JavaScript Code.....	200
Modify the Redirect Mechanism.....	200
Capture the Original Referrer.....	201
Verify the Referrer with the SiteCatalyst Debugger.....	201
Report to Variable Mapping.....	203
Variable to Report Mapping.....	208
External Email Tracking.....	213
Event Serialization.....	216
Methods of Event Serialization.....	216
Purchase Events.....	217
Adobe Mobile Tracking.....	219

Tagging Mobile Pages.....	219
How the Beacon Reaches Adobe.....	219
Network Protocols.....	220
Reports for Mobile Devices.....	220
Current Options to Improve Visitor Identification.....	220
Report Suite IDs - Dynamic Accounts.....	222
Common Errors.....	223
NS_Binding_Aborted in Packet Monitors.....	225
Contact and Legal Information.....	226

SiteCatalyst Implementation

This guide contains a description of SiteCatalyst data collection variables, and details on implementing SiteCatalyst data collection code in JavaScript.

AppMeasurement guides for other platforms and Mobile SDKs are available at

http://microsite.omniture.com/t2/help/en_US/home/index.html#Developer.



Note: The [Admin & Analytics Reference](#) is available. This help resource provides report and metric descriptions, as well as reference information for administrators of the Adobe Analytics.

Recent Updates

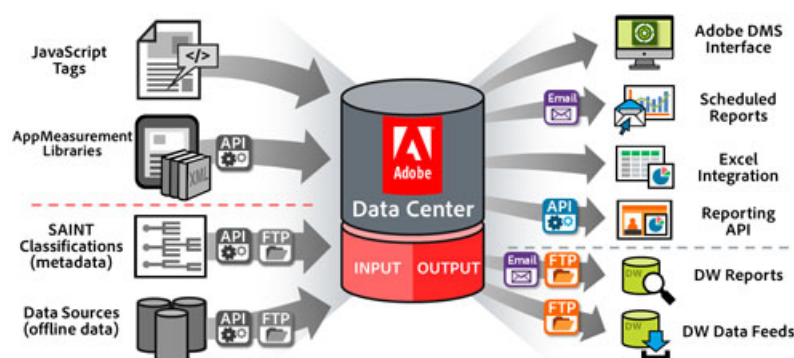
Date	Description
March 12 2013	Added details on decimal values in Currency/Numeric Events that are in the events list to events .
February 28 2013	Added details on how custom links affect Time Spent metrics to Tracking Rich Media - The t() and The tl() Functions .
February 21 2013	<ul style="list-style-type: none"> Added scope limitations for the <code>useForcedLinkTracking</code> option to Manual Link Tracking Using Custom Link Code. Updated pageURL to indicate that the size is no longer limited to 256 bytes.
November 15 2012	Added maxDelay variable information.
October 18 2012	Updated content in events . You can assign numeric integer values to counter events if you are on H.23 code or higher.
September 18 2012	Updated the Moving Data in and out of SiteCatalyst topic. (Previously called <i>Sending Data to SiteCatalyst</i> .)
September 6, 2012	Added List Var and List Prop topics.
August 30, 2012	Added several additional topics that were previously distributed as white papers: <ul style="list-style-type: none"> Report to Variable Mapping Page Naming External Email Tracking Event Serialization AJAX-Tracking Rich Media Applications Moving Data in and out of SiteCatalyst Adobe Mobile Tracking Common Syntax Mistakes in SiteCatalyst Report Suite IDs - Dynamic Accounts Collecting Data From Form Elements
July 19, 2012	Added information on custom link tracking on WebKit browsers introduced in H.25, including details on the overloaded <code>s.tl()</code> method. See Manual Link Tracking Using Custom Link Code .

Date	Description
April 20, 2012	<p>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.</p> <p>Added documentation stating that a single image request cannot contain a numeric or currency event higher than 2,147,483,648, or a counter event higher than 2,000,000,000. Note that this behavior has not changed, the limits are being documented for clarification.</p> <p>Added the <i>timestamp</i> variable.</p>
March 6, 2012	<p>Added a note to linkInternalFilters stating that you should 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.</p>
October 7, 2012	<p>Added <i>Cross-Device Visitor Identification</i>.</p>

Moving Data in and out of SiteCatalyst

Adobe has created multiple ways to send data into SiteCatalyst. These methods include tracking information in real-time from web sites, emails, campaigns, web-based kiosks, mobile devices, client-server applications, and most applications that can access the Internet.

These methods include reporting on information from offline systems via the Data Sources and partner integration platform. The image below provides a visual overview of the data collection and output options.



- [Input Options](#)
- [Output Options](#)

Input Options

JavaScript tags and AppMeasurement libraries are real-time, live data collection methods for SiteCatalyst. SAINT and Data Sources options represent post-collection methods for importing additional data into Adobe's analytics platform.

Input Option	Description
JavaScript Tags	<p>Today most digital analytics platforms use JavaScript tags to collect data from web sites and other web-based systems. This approach involves placing client-side JavaScript code within your web pages, which sends page, browser, and visitor data to SiteCatalyst. Most SiteCatalyst implementations leverage this web beacon approach.</p> <p>Increasingly, organizations use tag management solutions such as TagManager to deploy and manage their various JavaScript tags, including their SiteCatalyst page code. Another way of simplifying the deployment of JavaScript tag management is context variables and processing rules. This strategy gives the developer and the marketer more flexibility and control over their implementation.</p> <p>See JavaScript Implementation.</p>
AppMeasurement Libraries	<p>AppMeasurement libraries provide a mechanism for data collection when the JavaScript tag method isn't compatible with the device, application, or system to be tracked.</p> <p>Adobe has released AppMeasurement libraries for mobile devices (iOS, Android, Windows Phone), rich media (Flash-Flex, Silverlight), and other languages/frameworks (Java, .NET, PHP, XML). All of these libraries leverage the Data Insertion API to pass data to SiteCatalyst, which can also be used for batch uploads or delayed data collection.</p>

Input Option	Description
	<p>AppMeasurement libraries use cases include:</p> <ul style="list-style-type: none"> • Collect data on mobile apps and non-JavaScript supported electronic devices • Send order confirmation server-side from a back-end transactional system to reduce order count discrepancies created by client-side browser issues • Measure telephone call systems and intranets • Track actual file downloads rather than clicks on download links • Use as a workaround for the IE URL limitation (IE has a 2083 character limitation that can truncate your SiteCatalyst image requests if they exceed this threshold) <p>To link a visitor's behavior across JavaScript and tagless data collection, you may need to generate your own visitor ID (unique identifier for a particular visitor) so that the AppMeasurement and JavaScript values match. Without a tie to a common visitor ID, AppMeasurement data is typically collected in a separate report suite.</p> <p>For more information on the AppMeasurement libraries, visit Developer & Implementation.</p>
SAINT Classifications	<p>SAINT enables you to upload metadata for SiteCatalyst custom variables (prop or eVar). Metadata is data about data or attributes about your data. For a particular campaign tracking code, article ID, or product ID, you can assign multiple classifications to define or aggregate your data in various ways. For example, for each campaign tracking code you can specify the marketing channel, offer type, featured product, ad placement, creative type, and so on. Each classification generates its own unique report in SiteCatalyst, which can be broken down by other attributes or dimensions.</p> <p>SAINT classifications are retroactive. If you want to reclassify or change some metadata values, you can replace the SAINT file with an updated version. While many attributes are text-based (brand, color, size), a special form of classification (Numeric 2) allows you to create metrics (cost of goods sold) that can have different values for different time periods.</p> <p>See SAINT Classifications Home for more information.</p>
Data Sources	<p>In SiteCatalyst, you can import other forms of time-based data. Use Data Sources to upload offline metrics and to import data from other digital marketing systems, such as your email or ad serving vendor. In the upload process, you specify and map the metrics and data dimensions to particular custom events and eVars for reporting and analysis purposes.</p> <p>There are two main types of Data Source uploads. First, you can upload summary data, which attaches offline metrics to a subset of specified reports (eVars). Second, the Transaction ID method enables you to link an offline event to an online event through a unique transaction identifier that is set when an online event occurs, such as a lead form submission. The advantage of transaction ID data is that your uploaded metrics are fully integrated with the rest of your SiteCatalyst reports rather than a few designated reports with summary data uploads.</p> <p>Here are some common examples of how companies use Data Sources:</p>

Input Option	Description
	<ul style="list-style-type: none"> • Upload campaign metrics (ad impressions, campaign cost, etc.) via summary data tied to campaign tracking codes • Import cost of goods sold data via transaction ID to analyze gross margin performance • Tie product returns via transaction ID to close the gap between gross sales and net sales • Upload closed offline sales via transaction ID to understand what online tactics and behaviors are closing business deals, not just generating leads • Bring in financial loan approvals, declines, and loan value from offline processing via transaction ID • Import CRM data for data that may change over time, such as customer lifetime value, customer type (silver/gold/platinum), location, etc. via transaction ID <p>A third type of Data Source upload is known as <i>fully processed data</i>, which is used in limited cases for post-collection batch uploads (tied to visitor ID or IP/user agent). It is handled as if it were received by Adobe's data collection servers at the time specified. The uploaded data is indistinguishable from your normal SiteCatalyst data, and there is not a mechanism to delete the data if it is erroneous.</p> <p>For more information, see Data Sources.</p>

Output Options

The following table describes data output options.

Output Option	Description
Adobe Marketing Cloud Interface	<p>The most obvious way to access your SiteCatalyst data is through the Marketing Cloud interface. You can view and interact with your data in various ways and via different types of devices (desktop, tablet, mobile phone, or HD display). Data can be displayed, downloaded directly, or sent to other coworkers in various file formats, such as PDF, Excel, Word, and CSV.</p> <p>The Data Extract feature (found in most reports under More Actions > Extract Data) lets you customize a default report with various values for the X and Y axes. You can also bookmark custom reports and build interface-based dashboards. The menu structure and reports can be customized to meet your specific business needs. In addition, if you have Discover or Insight, you can perform deep, ad hoc analysis on your SiteCatalyst data in these advanced analytics tools.</p> <p>See Extracting Data from a Report for this procedure.</p>
Scheduled Reports	<p>In SiteCatalyst, you can schedule reports to be sent to different business users on a recurring basis. You can control the recipients, format, time frame (fixed or rolling), delivery frequency (daily, weekly, monthly, etc.), and so on.</p> <p>The publishing list feature enables you to distribute data from various report suites to different groups or sets of email addresses. For example, you could send out an external search report to multiple stakeholders, which would be based on their unique location/report suite (France vs. Canada). You also have the option of publishing a ReportBuilder (custom Excel-based) report rather than a SiteCatalyst report or</p>

Output Option	Description
	<p>dashboard. You also have alerts that can be configured to send out a report when a particular condition or threshold is met.</p>
Excel Integration	<p>You can pull data directly into Excel using the ReportBuilder add-in. ReportBuilder enables you to build customized data requests from SiteCatalyst, which can be inserted into your Excel worksheets and dynamically reference specific cells.</p> <p>Using ReportBuilder's data filters, you can extract and combine data points into highly customized reports. Once an advanced dashboard has been created, it can be manually refreshed or automatically published with the most up-to-date information.</p> <p>For more information see ReportBuilder Help.</p>
Reporting API	<p>If you would like to insert SiteCatalyst data and reports into third-party applications like an intranet or company-branded application, you can use the Reporting API for programmatic access to SiteCatalyst's core report data. Through the Reporting API, you can create customized reports that include calculated metrics and formatting options.</p> <p>SiteCatalyst 15, you can generate reports based on segments. The Reporting API lets you perform multilevel breakdowns across reports. See the API documentation within the https://developer.omniture.com for more information.</p> <p>While not as powerful or flexible as the Reporting API, the SiteCatalyst Widget is a simple way to insert SiteCatalyst reports into a web page, intranet page, or other HTML-based applications. This widget pulls in your bookmarked reports and dashboards from the SiteCatalyst interface into a designated location. You can download this widget tool from the SiteCatalyst interface.</p>
DataWarehouse Reports	<p>If you have DataWarehouse enabled for SiteCatalyst, you can generate ad hoc data reports on your historical SiteCatalyst data, which are delivered as CSV files via email or FTP.</p> <p>With DataWarehouse reports, you build a query to filter your SiteCatalyst data and isolate specific data points. The majority of requests take less than a day to process, but depending on the complexity of your query and the amount of data, it can take longer to process. DataWarehouse has a request manager interface and API to help manage and prioritize DataWarehouse reports.</p> <p>See DataWarehouse Requests.</p>
Clickstream Data Feeds	<p>You can combine clickstream data from SiteCatalyst with other data in an internal Enterprise Data Warehouse (EDW). The clickstream data feeds provide the stream of raw, partially processed server calls from SiteCatalyst in a delimited data set. The data feeds are configured at the report suite level and delivered on a daily basis (containing the hit data for the previous day). Depending on the average amount of data collected for the report suite, you can deliver it via FTP in a single zipped file or multiple zipped files (recommended).</p>

Output Option	Description
	For more information, see Clickstream Data Feeds .

Tracking Online Data

SiteCatalyst enables you to track various forms of online data, including Web pages, links and links as Web pages, page elements, and rich media. The process and syntax for tracking each of these online data formats are described in the following sections.

Tracking Web Pages

As a Web page with SiteCatalyst code loads, the SiteCatalyst code collects certain variables at the page and browser level and calls the SiteCatalyst JavaScript file.

These variables (and other identifiers) are used to facilitate the data collection process, and can be dynamically populated with server or application variables.

The JavaScript file builds an image request (web beacon). The image request uses a transparent 1x1 pixel image to pass data from the Web page to the Adobe data center. The process is repeated every time the visitor accesses a page that contains SiteCatalyst code. The following example shows a simple HTML page with SiteCatalyst code.

```

<!-- SiteCatalyst code version: H.xx.x.
Copyright 1996-2013 Adobe, Inc. All Rights Reserved
More info available at http://www.omniture.com -->
<script language="JavaScript" type="text/javascript"
src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/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>
<script language="JavaScript" type="text/javascript"><!--
if(navigator.appVersion.indexOf('MSIE')>=0)document.write(unescape('%3C')+'\!-!+-!')
//--></script><noscript></noscript><!--/DO NOT REMOVE/-->
<!-- End SiteCatalyst code version: H.xx.x. -->

```

A variety of mechanisms can be used to populate the variables, including server-side code, JavaScript functions, or event plug-ins within the SiteCatalyst JavaScript file. The following example shows a custom plug-in residing in the JavaScript file that populates the *campaign* variable based on query string parameters in the URL.

```
function s_doPlugins(s) {  
  /* a,b,c = list of query string parameters to look for and  
  populate the "s.campaign" variable. */  
  s.campaign=s.getQueryParam("a,b,c")  
}
```

Tracking Links and Page Elements

In addition to tracking the page load, SiteCatalyst also tracks specific actions on a page.

The SiteCatalyst JavaScript file attaches a listener function to the *onClick* event (using either the *attachEvent* method for Internet Explorer browsers or the *addEventListener* method for Netscape/Mozilla browser) in the document body. This enables ClickMap, exit link tracking, download link tracking, form analysis, and the use of the *link handler* plug-ins.

Specific business requirements can be met by passing in SiteCatalyst variables on link clicks. Additionally, custom link tracking tracks additional information about a link when the standard methods are not sufficient. More detail on each of these is provided below.

ClickMap

ClickMap is a browser plug-in, and a module of SiteCatalyst, that lets Adobe clients visually measure traffic (page views), conversion, and success metrics within the pages of a website. These metrics are overlaid on top of the page's links and answer critical questions, including the following:

- What is the value of an individual page on your site?
- What is the value of an individual element on a Web page?
- What is the most valuable digital real estate on your page?

A visual display highlights the most relevant elements on your Web page and calculates the overall value of that page to your site's success, based on any other success metric you define. **ClickMap** provides all this data by tracking the links clicked on a page.

Exit Link and Download Link Tracking

The **Exit Link Report** shows links a visitor clicked to leave your site and go to another site. The **Exit Link Report** shows links a visitor clicked to leave your site and go to another site. The **Download Link Report** helps you understand how often your visitors download files from your site. SiteCatalyst automatically tracks file downloads and **Exit** links based on configuration parameters set in the JavaScript file.

For exit link tracking, any click of a link that does not match a list of internal URL filters generates an image request to SiteCatalyst. For download link tracking, any click on a link that matches a list of download file types generates an image request to SiteCatalyst. Links that need additional information passed about them can be manually tracked using custom link code as explained in the [Tracking Custom Links](#) section.

Link Handler Plug-in

The *linkHandler* plug-in was created to automate custom link tracking and conditionally track custom links. The *linkHandler* plug-in takes advantage of the fact that the SiteCatalyst JavaScript file attaches to the *onClick* event. Instead of manually adding functionality to the *onClick* event of a link (see [Tracking Custom Links](#)), the *linkHandler* plug-in identifies links, sets a link type (Exit or Download), sets variables, or fires events whenever a link is clicked. Specific examples include the following:

- Identify links that go thru a redirect and should be considered exit links, even though they match the *linkInternalFilters* variable.
- Set an event or SiteCatalyst variable on a link click.
- Give friendly names to file downloads based on filters.
- Copy the URL of the link clicked into a prop variable to allow for classification on those values.
- Reporting exit, download, or custom links as page views in SiteCatalyst.

Tracking Custom Links

Custom link code lets file downloads, exit links and custom links be tracked in situations where standard link tracking is not sufficient or applicable. Custom link code is typically implemented by adding an *onClick* routine to a link, or adding code to an existing routine. However, it can be implemented from any JavaScript event handler or function.

The basic code to track a link using custom link code is shown in the following example:

```
<a href="index.html" onClick="var s=s_gi(report_suite_id); s.tl(this,'o','Link Name'); ">My Page</a>
```

The text `report_suite_id` must be replaced with the appropriate **report suite ID**. Note that the *s_account* variable has been initialized and can be used in this function. This example is for H-code.

Check with your Implementation Consultant if you need G-code. See the [Link Tracking](#) white paper for more information.

Tracking Rich Media - The t() and The tl() Functions

The standard Adobe JavaScript code contains two functions used to track user interaction in a non-HTML environment such as AJAX.

The standard Adobe JavaScript code contains two functions used to track user interaction in a non-HTML environment such as AJAX. These functions, *t()* and *tl()*, have different properties and uses.

Function	Description
<i>t()</i>	This function is used to track a virtual web page load. Calling this function results in a page view.
<i>tl()</i>	This function is used to track links. Calling this function results in a custom link, download link, or exit link .

These functions are for H-code. Check with your Implementation Consultant if you need G-code. Each time the *t()* function is called, SiteCatalyst increments the total page views for the site. Additionally, any variables (Adobe JavaScript variables) that have a value (other than an empty string: " ") at the time the function is called, are sent to SiteCatalyst. Use *t()* when an event should be considered a page view, for example, when moving from one page to another in a multi-page AJAX application. Calling *t()* adds another page to the path reports (**Next Page Flow**, **Fallout**, and so forth), as well as impact the time-spent-on-page calculations.

The previous section shows that *t()* sends data in the same way that a page load does. The *tl()* function, however, sends data in the same way that custom link tracking does. The total page views to the site are not incremented when calling *tl()*, but non-page view data is recorded. Calling *tl()* does not add another path view to pathing reports for **Pages** (**Next Page Flow**, **Fallout**, and so on). However, path views for variables included in *linkTrackVars* is calculated if pathing is enabled for those variables.

Page Views vs Links

One of the most commonly asked questions with rich media 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 lets customers uniquely configure a product. The application may have significant steps 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 rich-media is that each application is unique. They are designed to mimic realistic actions and because 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. While each application requires some consideration as to what specifically should be measured, there are some generalizations that can be applied to rich-media tracking.

Activity	Description
Macro-level Activity	This usually constitutes the loading of the application, which provides information on visits, visitors, instances, value to future actions, and so forth. It should also represent major steps in the process. If a rich media 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	<p>This includes any changes less than 50% (or not considered as significantly changing the user experience or content). Toggling between color selections, for example, is considered micro-level activity.</p> <p>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. When measuring the effectiveness of rich media, 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. This tracking ensures that page views are not inflated by micro-level activity, and allows for path analysis through the traffic variable.</p>

ActionSource

Adobe ActionSource is a patent-pending method for natively tracking Flash applications used with the ActionScript programming language.

This solution removes dependencies on JavaScript for Flash tracking through the use of native ActionScript components.

Historically, tracking Flash has been dependent on two programming languages: ActionScript and JavaScript. Communication between these technologies complicated the implementation process and introduced technical barriers. For Flash developers who did not understand the nuances of JavaScript, it was difficult to put analytics solutions in place. Even for those who could deploy the JavaScript solution, limitations based on language and browser communication caused performance and data integrity problems including the following:

- Cancellation of analytics when Flash attempted to communicate through the browser to JavaScript for analytics and to link out to another browser page.
- Limitations of 508 characters per data transmission from Flash through the browser to JavaScript.
- Playback delays for animation while JavaScript is executing.
- Localized tracking only - JavaScript must be present on the same page as the Flash file to execute analytics tracking.

Adobe ActionSource was developed to simplify the implementation process and to improve performance and accuracy. Its native ActionScript engine removes the dependency on JavaScript and maximizes application portability and accuracy. Some of the key benefits of this solution are listed in the table below.

Media Tracking

Media on your site is tracked by gathering basic information from the media player and building a session of events that are sent to the Adobe collection servers for processing.

The following basic information is either collected automatically or provided to ActionSource or JavaScript:

- Media Name
- Media Length (in seconds)
- Media Player Name

Both ActionSource for AS3 and the Adobe JavaScript collection file version H.15 and up support the Media Tracking module. The media module is the option when using JavaScript, but is always present in ActionSource. If "s" is the name of your JavaScript object or ActionSource instance, you reference the media module as *s.Media*. In both implementations you manually track a media session by calling four functions. Alternatively, media can be automatically tracked by setting the *s.Media.autoTrack* variable to true.

Media tracking is accomplished by collecting data about which portions of the video are viewed and sending that data at the end of the video. Because your media player or browser window may be closed before the media has finished playing, the collection code buffers the media session in a cookie or flash shared object that is passed in with the next page view. The cookie and shared object are named *s_br*. You can disable the cookie or Flash shared object by setting the *s.disableBufferedRequests* variable to true. This cookie and shared object are retained for 30 minutes because the data must be associated with the visit in which it occurred.

Tracking Links as Web Pages

There may be times when you want to report a link clicked as a page view.

For example, you want to add the link to the path reports or more accurately report time-spent-on-page metrics. There are two alternatives to count links as page views.

Links that Leave the Page

If the link leaves the page, simply use standard link tracking (exit link and download link) or custom links, or an explicit call to *s.tl()*. Each of these functions ensures that the image request is fired before the user leaves the page by inserting a 500 ms delay. VISTA can then be used to convert these links into page views.

Links that Do Not Leave the Page

Alternatively, you may have links that simply change elements on a page, for example DHTML or open a new page in a new window. In these cases, the 500ms delay is not needed, since the page doesn't change. To record these links as page views, use the *linkHandler* plug-in. Contact your Implementation Consultant for configuration instructions.

For rich media applications, call *s.t()* directly. To record links as links without incurring the 500ms delay, a VISTA rule can be written to convert a page view (sent via *s.t()* or the *linkHandler* plug-in) into a custom, download, or exit link.

Using the Data Insertion API

The XML Data Insertion API is a tool that facilitates sending page views into the SiteCatalyst Data Processing Engine via an XML request (instead of standard Web beacon data collection).

The XML Data Insertion API is a tool that facilitates sending page views into the SiteCatalyst Data Processing Engine via an XML request (instead of standard Web beacon data collection). It has been designed to facilitate the integration of SiteCatalyst with applications that cannot be easily tagged with JavaScript or when a browser is not used.

Examples include the following:

- Orders and pages where the URL of the image exceeds 3kb.
- Actual file downloads, rather than clicks on links. For example, direct links from Google to non-HTML files.
- Order confirmation from a back-end system.
 - Orders are verified up to 30 minutes after purchase due to fraud or credit checks.
 - Any confirmation that happens outside of the website, like email confirmation.

Tracking Offline Data

SiteCatalyst lets you track various forms of off-network and third-party applications with Data Sources and various partner integration formats.

Tracking Off-Network and Third-Party Applications - SiteCatalyst Data Sources & Partner Integration

Customers have requested the ability to import other types of time-based data to leverage the features of SiteCatalyst. Examples of data that can be imported via data sources including the following items:

- Email or ad vendor data such as sends, opens, and impressions
- Off-line sales
- Off-line returns

SiteCatalyst integrates with other data providers, including:

- DoubleClick for integrating email and ad-server integration
- Salesforce.com for integrating CRM/SFA data
- LinkShare for affiliate marketing
- Advertising.com for integrating ad network data
- Vignette for content management
- CheetahMail for email remarketing
- Offermatica for A/B testing, multivariate analysis, and optimization
- OpinionLab for user experience and feedback data
- Various others

Tracking Additional Attributes - SiteCatalyst Classifications

To make your analytics data more actionable, SiteCatalyst provides customers with a powerful classification system. The classification system allows you to associate SiteCatalyst data with additional attribute dimensions. The Adobe SiteCatalyst classification system can be configured to pivot around any SiteCatalyst variable including custom insight variables, custom conversion variables, *campaign* variable, domains, zip codes, and many others.

Here are examples of using classifications.

1. Classifying campaign tracking codes with:
 - a. Creative type (such as link, image, or keyword)
 - b. Creative content (like an image of a father and daughter or an email to a friend)
 - c. Campaign type (such as email, banner, or pay-per-click)
 - d. Campaign vendor (such as CheetahMail, DART, Google, or Overture).
 - e. Cost
2. Classifying productID with:
 - a. Manufacturer

- b. Size
 - c. Color
- 3. Classifying ad IDs with:
 - a. Advertiser company
 - b. Advertiser content
 - c. CPC

JavaScript Implementation

To begin using SiteCatalyst, data must be sent to a report suite to display in reporting. The easiest and most common way to send data to SiteCatalyst is by using JavaScript implementation.

To successfully implement a page with code to collect data, it is helpful to 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 provide instructions on how to do this, even for an empty web page.

Caching

The JavaScript file is cached in the visitor's browser after it is initially loaded. The file is downloaded only 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. Transferring most of the JavaScript into this file results in less overall downloaded data.

SiteCatalyst JavaScript Compression

If you are concerned about page weight (size), 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 SiteCatalyst JavaScript file.

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

- http://httpd.apache.org/docs/2.0/mod/mod_deflate.html
- <http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/25d2170b-09c0-45fd-8da4-898cf9a7d568.mspx?mfr=true>
- <http://www.cubicleman.com/2007/04/06/enabling-gzip-compression-with-tomcat-and-flex/>

How does SiteCatalyst Collect and Report Data?

The SiteCatalyst code (or Code to Paste) is placed in the HTML source code on the desired pages of your production websites.

When visitors enter your website, your landing page loads in their browser windows. As the page loads, the SiteCatalyst **Code to Paste** sets certain variables. For example, the *pageName* variable, and the SiteCatalyst JavaScript Include file. The variables (and other identifiers) facilitate the data collection process and can be dynamically populated with server or application variables.

The JavaScript Include file builds an image request, or web beacon. The image request uses a transparent 1x1 pixel image to pass data from the Web page to the Adobe data center. No personal data about visitors is passed to Adobe. The process is repeated every time the visitor accesses a page on your site that contains SiteCatalyst code.

The image request contains a query string that passes the data variables that are collected on the page. Some of these variables are set specifically in the **Code to Paste**, and some variables are set automatically within the JavaScript file. Additionally, the image request contains a number of variables in the HTTP header. All of these variables constitute the sum of the data collection elements.

When the HTML **Code to Paste** and JavaScript files are in place, and the Adobe consultant has made the appropriate configurations, the Web analysis data for your website is stored in tables. These tables are called report suites. The reports in the SiteCatalyst interface draw their data from the report suite tables.

Step 1 - Create a New Report Suite

Create a report suite so SiteCatalyst knows where to process and store your data.

1. Login to SiteCatalyst and click **Admin > Admin Console > Report Suites**.
2. Click **Create New > Report Suite**.
3. Leave template set to **Default**.
4. Input the **Report Suite ID** (Abbreviated RSID).

This is how our database and your page code identifies your report suite. It can be any string of alpha-numeric characters with a static prefix that cannot be changed. Since this is going to be a test report suite, place test or dev at the end of the RSID so it doesn't get mistaken for actual data. It is not required to input the prefix in the RSID field. SiteCatalyst attaches the prefix for you. If you type the prefix in the RSID, you end up with a report suite ID that contains a duplicate prefix.

5. Input the **Site Title**.

This is the friendly name, or what you see in the SiteCatalyst interface and in reports.

6. Input the **Time Zone** and **Go Live Date**.
7. Input the **Estimated Page Views per Day**.

This is required so our network operations team knows approximately how much resources they want to put towards this report suite.

8. Click **Create Report Suite**.

Step 2 - Generate Code

SiteCatalyst has a place to store your data for this website. When Step 1 is completed, you can obtain the code that is going to be implemented on your page.

1. Go to **Admin > Admin Console > Code Manager**.
2. Under **Select the Type of Code to Generate** select **JavaScript**.
3. Select the newly created report suite.
4. Select the Character-Encoding of your website.

It is usually near the top of your HTML code. If you are unsure, use **UTF-8**.

5. Select the number of periods in your domain name.
6. Click **Generate Code**. Read the warning message, then click **OK**.

The resulting page contains two parts of code: **Page Code** and **Core Javascript File**. Keep this page open, as it is referenced several times.

Step 3 - Modify Your Web Page

These steps help you modify your Web page.

1. Open your Web page in an HTML editor.

If creating a testing Web page, create a new HTML site. The following code looks familiar if you have worked in Web page programming before:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Testing SiteCatalyst Implementation</title>
```

```
</head>
<body></body>
</html>
```

2. Paste the page code into the <body> tag of the HTML.

Do not place SiteCatalyst code in the <head> tag . If you are more concerned with data accuracy, put the code as close to the top of the <body> tag as possible. If you are more concerned with loading the page before any data is sent, place the code toward the bottom of the <body> tag.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Testing SiteCatalyst Implementation</title>
</head>
<body>
<!-- SiteCatalyst code version: H.xx.x. Copyright Adobe, Inc. All Rights Reserved. -->
<script language="JavaScript" type="text/javascript"
src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/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.prop1=""
s.campaign="" s.events=""
s.products="" s.eVar1=""
/***** DO NOT ALTER ANYTHING BELOW THIS LINE ! *****/
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>
<!--/DO NOT REMOVE/-->
<!-- End SiteCatalyst code version: H.xx.x. -->
<!-- This short paragraph was also inserted so the page would not be completely blank after
opening it -->
<p>This is a practice page to test SiteCatalyst and learn how to successfully implement
it.</p> </body>
```

3. Create a new JavaScript file and name it s_code . js.

Paste the entire Core JavaScript File code within the file. No contents need to be altered.

4. On the **Page Code**, change your code to point to your s_code . js file.

For example, if you are implementing this code on adobe.com and are going to host these files in the root directory:

Take this line of code:

```
<script language="JavaScript" type="text/javascript"
src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/s_code.js"></script>
<script language="JavaScript" type="text/javascript" src="s_code.js"></script>
```

And direct it to where you upload your core javascript file:

```
<script language="JavaScript" type="text/javascript"
src="http://www.adobe.com/s_code.js"></script>
```

Alternatively if your page code and s_code.js file is in the same location (e.g. on the desktop or the same FTP folder):

```
<script language="JavaScript" type="text/javascript" src="s_code.js"></script>
```

5. Alter the **s.pageName** variable to Test Page:

```
s.pageName="Test Page";
```

Your **Core JavaScript File** and **Page Code** are ready. Save and upload the files to your hosting servers to be accessed from a Web browser. If you do not have access to any hosting servers, you can save them to your hard drive. Make sure the uploaded location matches the domain and path-to-code specified above.

Step 4 - Validate Your Implementation

Ensure that everything is set up correctly on the site and ensure it appears in reporting.

1. Obtain DigitalPulse Debugger, which lets you see SiteCatalyst variables while visiting your website.
2. Navigate to your new Web page, or open the saved HTML file on your hard drive. Open DigitalPulse Debugger.
3. Review the above steps to ensure everything was done correctly and try again.

The *pageName* parameter is **Test Page**. If you see a window that says **No requests found**, it means there is an issue with your implementation.

4. Once verified in DigitalPulse Debugger, wait several hours to give the processing servers time to populate SiteCatalyst reports.
5. Login to SiteCatalyst. Go to **Site Content > Pages**.
6. Look for **Test Page** with one or more page views.

If you see page views in the report, you have successfully implemented SiteCatalyst on a page. When implementing on multiple pages, the **Page Code** must be present on each, but can reference the same core Javascript file. Each page can also have their own variables such as **props**, **eVars**, *products*, **campaigns**, and so forth.

If you require assistance implementing code on your site, contact your organization's account manager who can arrange a meeting with an Adobe Implementation Consultant.

Tracking Across Different Implementation Types

This information is intended for advanced SiteCatalyst 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:** Depending on your data center and location, these can greatly vary. Have one of your organization's supported users contact Adobe ClientCare if you are not sure which one to use:

Non-RDC domains

- San Jose: `112.2o7.net`
- Dallas: `122.2o7.net`

RDC domains

- San Jose: `d1.sc.omtrdc.net`
- Dallas: `d2.sc.omtrdc.net`
- London: `d3.sc.omtrdc.net`

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 **visid** 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";
```

Collecting Data From Form Elements

SiteCatalyst lets you 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 SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst JavaScript 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 SiteCatalyst 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.

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.

RIA looks cool, and is changing the Web, but, should you use it on your site? This is an area where caution is important. Rich Internet Applications are expensive to develop. Don't jump in without using SiteCatalyst to test the water.

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 SiteCatalyst 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

Implementing 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.

Designing 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.

Deploying the Code

There are two functions in the JavaScript code that allow you to send data to SiteCatalyst. There are some distinct guidelines that should be followed to know which method should be used to send data.

Collecting 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.

Sending Macro-Data (Page)

If an image request was previously made on the same page, you can first clear the values of the previously-set variables. Write a simple JavaScript function to clear the Adobe variables. Set the values appropriate for the changed content, namely the *pageName* variable. After the variables are set call the *t()* function.

Syntax

```
//clear and set variables
void(s.t());
```

Example

```
s.pageName="New Page"
s.prop1="some value"
void(s.t());
```

Collecting 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.

Sending 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

```
s=s_gi('rsid[,rsid2]');
//set linkTrackVars and linkTrackEvents> (if applicable)
//set new variables>
s.tl(this,'o','Link Name');
```

Example

```
s=s_gi('myreportsuiteid');
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.

Implementing SiteCatalyst without JavaScript

SiteCatalyst data collection is usually implemented using an HTML image tag that is created using JavaScript.

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	<p>This file contains common code that:</p> <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 SiteCatalyst 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 https: (secure) connections, or 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 ClickMap, download links, exit links, and browser-based variables (browser width/height, etc.) cannot be collected or supported using this implementation method.

PHP Measurement Library

Adobe offers a PHP measurement library containing a PHP class you can implement on your Web server to dynamically build non-JavaScript image requests.

This class allows you to populate any SiteCatalyst variables and manage visitor IDs, server-side (to some extent). This solution automates the building of the image tag described in this section, making it easy to get started on a non-JavaScript implementation using PHP.

You can download the necessary PHP code from the **Code Manager** in the SiteCatalyst Admin Console.

Variable Names

The names of SiteCatalyst variables differ between the image request and the JavaScript variable name.

This guide uses the JavaScript variable name exclusively. The following table maps the JavaScript variable to the query string parameter name in the SiteCatalyst image request.

JavaScript Variable	Query String Parameter
s.pageName	<i>pageName</i>
s.server	<i>server</i>
s.pageType	<i>pageType</i>
s.channel	<i>ch</i>
s.prop1 - s.prop75	<i>c1 through c75</i>
s.campaign	<i>v0</i>
s.state	<i>state</i>
s.zip	<i>zip</i>
s.events	<i>events</i>
s.products	<i>products</i>
s.purchaseID	<i>purchaseID</i>
s.eVar1 - s.eVar75	<i>v1 through v75</i>
*Link Type	<i>pe (lnk_d, lnk_e, lnk_o)</i>
*Link URL	<i>pev2</i>
Referring URL	<i>r</i>
Current URL	<i>g</i>



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.

With server-generated image tags, the image request is executed by a Web browser. If you need to send data directly from your servers to Adobe's servers, contact Adobe **ClientCare** to determine the best transfer method.

Other Requirements

There are additional requirements and configurations for implementing SiteCatalyst 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 <i>Variable Names</i> in this document.												
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 <i>SiteCatalyst Variables</i> . Exceeding the maximum length for a variable causes the value of this variable to be truncated for storage and display in SiteCatalyst.												
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> <tr> <th>Protocol</th><th>URL</th></tr> <tr> <td>https:</td><td>https://namespace.<data center-specific...>.2o7.net/b/ss/<u>reportsuite</u> /1/G.5--NS/...</td></tr> <tr> <td>http:</td><td>http://namespace.<data center-specific URL*>.2O7.net/b/ss/ <u>reportsuite</u> /1/G.5--NS/...</td></tr> </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 SiteCatalyst 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> <table> <tr> <th>Protocol</th><th>URL</th></tr> <tr> <td>https:</td><td>https://102.112.2O7.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...</td></tr> <tr> <td>http:</td><td>http:// <u>suite1</u> .112.2O7.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...</td></tr> </table>	Protocol	URL	https:	https://namespace.<data center-specific...>.2o7.net/b/ss/ <u>reportsuite</u> /1/G.5--NS/...	http:	http://namespace.<data center-specific URL*>.2O7.net/b/ss/ <u>reportsuite</u> /1/G.5--NS/...	Protocol	URL	https:	https://102.112.2O7.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...	http:	http:// <u>suite1</u> .112.2O7.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...
Protocol	URL												
https:	https://namespace.<data center-specific...>.2o7.net/b/ss/ <u>reportsuite</u> /1/G.5--NS/...												
http:	http://namespace.<data center-specific URL*>.2O7.net/b/ss/ <u>reportsuite</u> /1/G.5--NS/...												
Protocol	URL												
https:	https://102.112.2O7.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...												
http:	http:// <u>suite1</u> .112.2O7.net/b/ss/ <u>suite1,suite2</u> /1/G.5--NS/...												

Requirement	Description
	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.
URL and Referring URL	<p>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>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>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>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>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>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.sample.com/page.asp?parameter=1</code>) or certain file extensions (<code>.asp</code>, <code>.jsp</code>, etc.) are also not cached.</p> <p>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>
nameSpace Variable	<p>The nameSpace query string parameter is required for non-JavaScript implementations.</p> <p>Example: <code>ns=nameSpace</code></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>
```

Supported SiteCatalyst Reports

Most of the out-of-the-box SiteCatalyst reports are available with a non-JavaScript implementation.

Some of the reports do require JavaScript in order to show data. For example, the **Browser Height** report takes JavaScript information from the browser. Therefore, if you decide you do not want to implement with JavaScript, you will not have every SiteCatalyst report available for use.

You can work with Adobe ClientCare to use other variables to populate those reports. For more information, contact Adobe ClientCare.

Implementing SiteCatalyst on Mobile Sites

People are increasingly using mobile devices, such as cell phones, personal data assistants, and portable media players to access information via the Internet.

Because of the increasing use of mobile devices, Adobe gives you the option of tracking mobile users in SiteCatalyst, **Discover**, DataWarehouse, and other Adobe applications to effectively monitor those accessing your websites using mobile devices.

Because most mobile devices do not currently support JavaScript, the standard JavaScript tracking beacon cannot be used. To track mobile device users, you need to implement an alternative tracking method on your Web pages. You can use JavaScript and the mobile tracking method concurrently.

Because many mobile devices do not execute JavaScript, the Adobe best practice is to deploy a hard-coded tracking beacon for mobile sites. For information on configuring SiteCatalyst to report mobile devices, see [Implementing without JavaScript](#).

The following table contains the resulting URL parameters you can populate:

Query String Parameter	JavaScript Variable Equivalent
gn	s.pageName
sv	s.server
gt	s.pageType
ch	s.channel
c1 - c75	s.prop1 - s.prop75
h1 - h5	s.hier1 - s.hier5
v0	s.campaign
state	s.state
zip	s.zip
ev	s.events
pl	s.products
pi	s.purchaseID
v1 - v75	s.eVar1 - s.eVar75
ce	s.charSet
cc	s.currencyCode
pe (lnk_d, lnk_e, lnk_o)	Link Type
pev1	Link URL
pev2	Link Name
pev3	Video Reports
r	Referring URL
g	Current URL
D	s.dynamicVariablePrefix
cdp	s.cookieDomainPeriods

Query String Parameter	JavaScript Variable Equivalent
cl	s.cookieLifetime (s_vi cookie lifetime in seconds)
/5/ or /1/	s.mobile

Video Measurement

You can track media on your site by gathering basic information from the media player and building a session of events that are sent to Adobe's collection servers for processing.



Note: *The Media module received significant updates with the release of SiteCatalyst 15. For a complete Video Implementation guide across all supported platforms (JavaScript, Flash, Silverlight, iOS, and others), see [Measuring Video in SiteCatalyst](#).*

Processing Rules

Processing rules let you make changes to SiteCatalyst data as it is received.

For more information, see [Processing Rules](#) in the *Analytics Reference*.

Other AppMeasurement Libraries

Link to a full list of AppMeasurement libraries.

http://microsite.omniture.com/t2/help/en_US/home/index.html#Developer

TagManager

TagManager is the standard method for deployment of Adobe Marketing Cloud tags.

TagManager lets you:

- Deploy Marketing Cloud products and Genesis integrations through a single hosted JavaScript file.
- Manage tags and configuration in a secure administrative interface.
- Test changes and quickly roll back to previous versions.
- Use existing SiteCatalyst tags to implement TagManager without a complex deployment process.
- Deploy third-party tags.

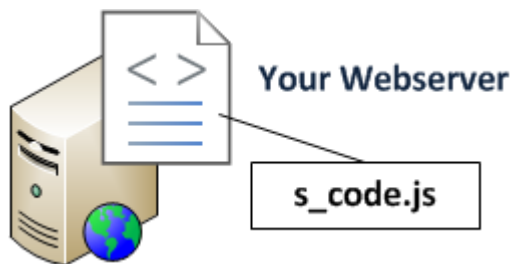
TagManager makes you less dependent on the IT update cycle to enable new marketing tools on your site. You can add new products, deploy third-party tags, and make other updates without changing tag code on your Web pages.

How TagManager Works

An overview of TagManager.

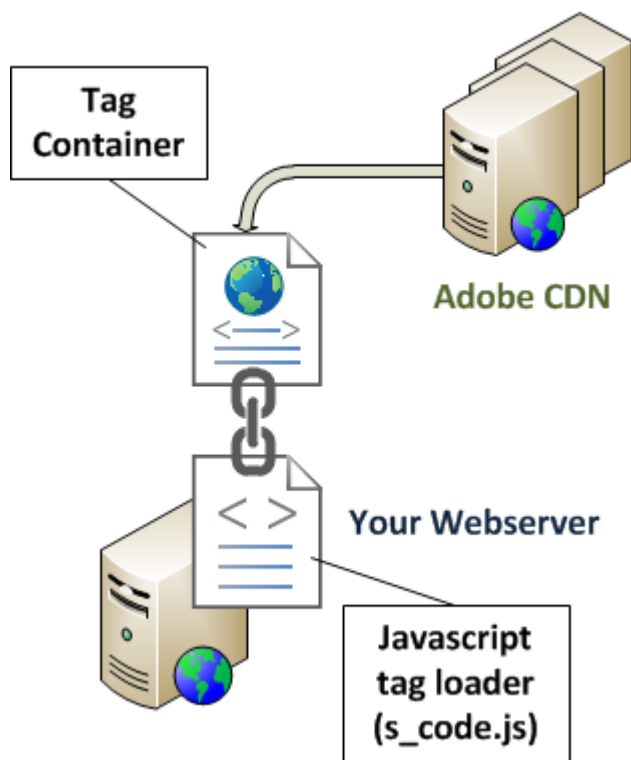
Measuring anonymous audience data is performed by placing a tag on the pages on your site. This tag typically references a static Javascript library that provides the underlying functionality to send data to a collection server. Depending on the products you have implemented, multiple tags and libraries are often required on your site, and updating tags to deploy a new product can be time consuming and error prone.

A standard SiteCatalyst implementation includes a Javascript file that you host your site, called `s_code.js`. This file provides the core functionality to send data to Adobe collection servers. When updates are released or if you want to use an additional module or plug-in, this file must be updated.

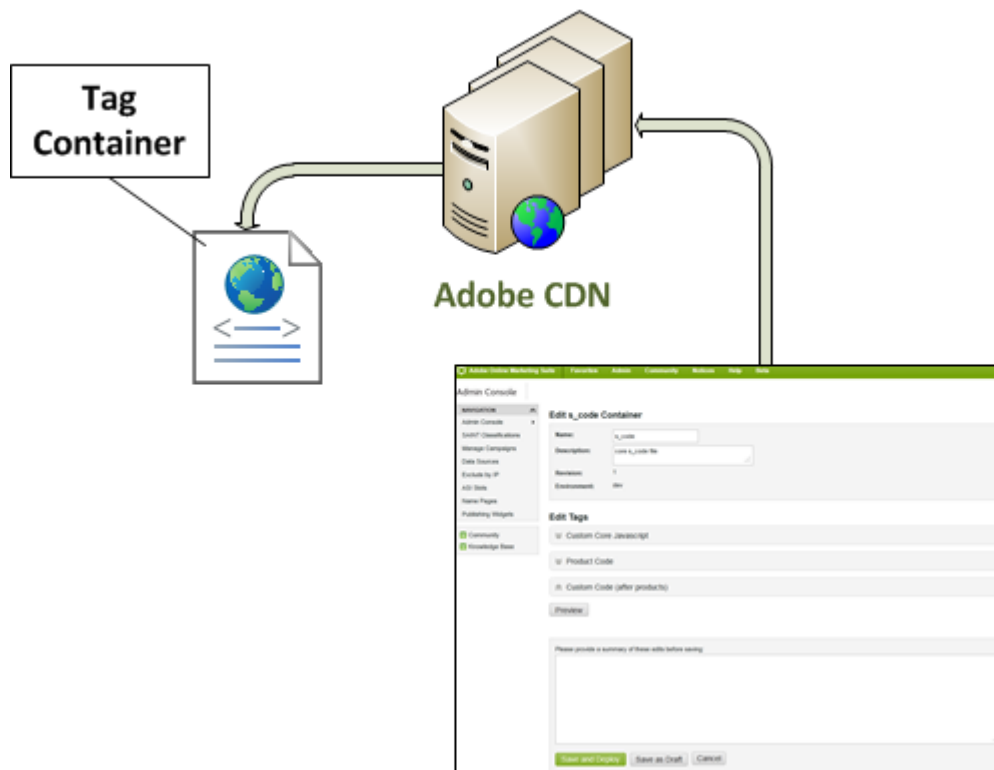


TagManager

TagManager simplifies tag deployment and updates by providing a tag management framework. When using this framework, instead of hosting the measurement code directly, you include a Javascript loader file that downloads and includes a tag container. The tag container downloaded by the loader is hosted for you by Adobe on a CDN. The hosted tag container provides the core functionality that was previously present in your static Javascript file:



This tag container is created and managed using the Marketing Cloud interface. Using the interface you can quickly deploy additional Marketing Cloud tags and Genesis integrations:



Tag Container Environments

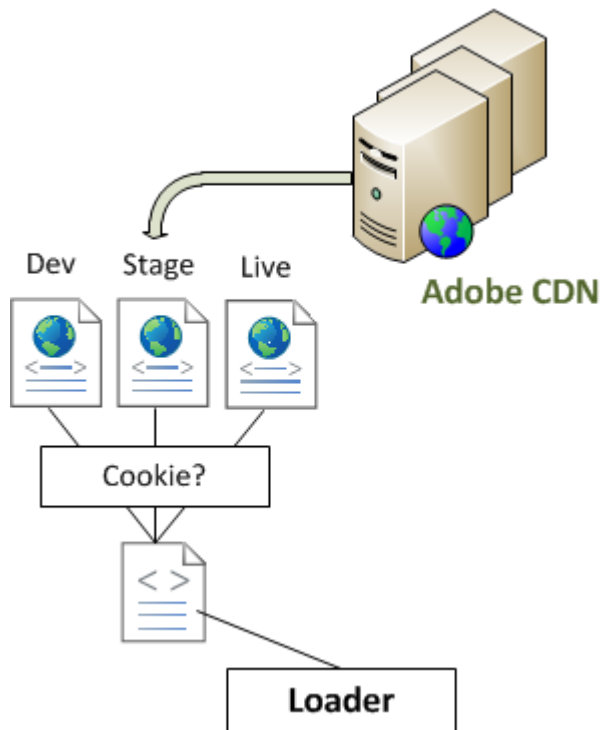
TagManager provides a dev, stage, and live environment for each tag container.

Each environment has a separate current version and revision history.

TagManager uses the same Javascript tag loader for each environment. This lets you test multiple code versions on the same site without requiring page-level changes. You can use these environments to align with your existing release process. For example, your development team can use the dev version of the tag container to test changes directly on your live site without affecting the live version.

Selecting the Tag Container Environment

When you load a page that includes the tag loader, an attempt is made to load the current live version of the tag container. To instruct the tag loader to retrieve the version from the stage or dev environment, you can set a cookie in your browser that instructs the JavaScript tag loader to retrieve a different version:



The cookie can be set using the bookmarklets available in the TagManager interface. If your browser does not support adding the bookmarklet, you can manually create a bookmark by copying and pasting the URL:

```
javascript:document.cookie='s_tagEnv=dev;%20path=/';void(0)
```

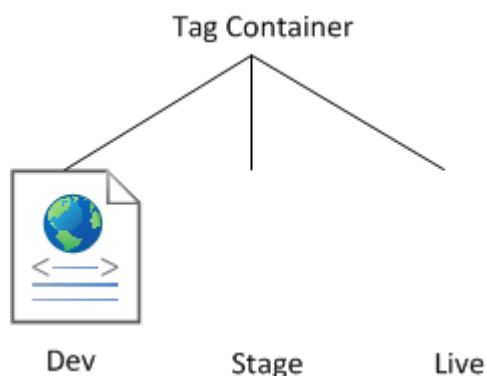
These bookmarklets can be used to quickly switch between the current version in each environment.

Setting the Dev Cookie to Test a New Tag Container

When you create a tag container, a version is initially created in the dev environment only:

Dev Environment				File History
File Revision	Last Edited	Comments	Status	Actions
Revision 1	by bwatson on 06/22/11 at 10:03:49am	Added SiteCatalyst, Test&Target, Demdex, Survey, Facebook integration and custom evar setting for new accounts.	deployed on 09/09/11 at 11:06:15am	Edit Copy to Stage

At this point, there is no file in the stage or production environments.



If you browse to a site that includes the tag loader without setting the dev cookie, an attempt is made to load the live version. Since there isn't a version in live, the tag container is not found and the loader fails gracefully.

If a tag container fails to load make sure the dev cookie is set correctly.

Asynchronous Loading

Tag container code is loaded asynchronously.

This provides a number of performance benefits, since your page continues to load while product tags execute. Because there isn't a guaranteed execution order between variables set on the page and variables set in a tag, you might need to make some adjustments to your code.

To avoid conflicts between page code and tag container code, use conditional statements or place calls in the `doPlugins` method which is executed after page code executes.

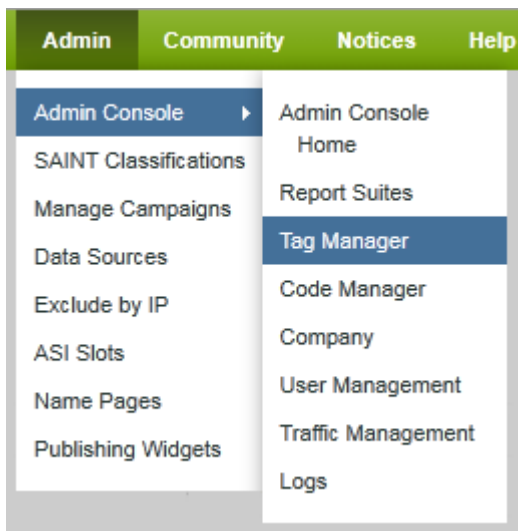
TagManager Quick Start

This section contains instructions to deploy TagManager with a new implementation.

Step	Task	Location	Description
Step 1	Create a New Container	TagManager UI	Define a tag container to provide analytics on your website.
Step 2	Define Tags	TagManager UI	Add support for products, Genesis integrations, and affiliate beacons to the tag container.
Step 3	Copy Environment Bookmarklets	TagManager UI	Save bookmarklets that let you switch between the dev, state, and live environments to test your tag container.
Step 4	Copy the Tag Loader to your Website	Your website	Each tag container is loaded using a unique version of the Javascript tag loader. Host a copy of this file on your website.
Step 5	Include the Loader on Site Pages	Your website	Add a <code><script></code> tag to each page on your website to include the Javascript tag loader.
Step 6	Test and Promote Revisions	TagManager UI	Set up your workflow to test and promote tag container versions.

Create a New Container

The first step is to define a tag container in the Marketing Cloud Admin interface.



On the **TagManager** page, click the **Create New Container** button.



Note: *Tag Containers cannot be deleted or renamed after they are created.*

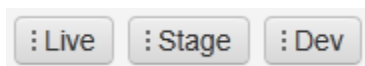
Define Tags

After you create a new tag container, the **Edit Tags** page displays. This page is where you select the core functionality provided by the tag container and add any custom code.

If you are existing SiteCatalyst customer migrating to TagManager, see [Migrating to a Tag Container](#).

Copy Environment Bookmarklets

To test each version of the tag as it moves from dev, stage, and live, copy the bookmarklets from the **Environment Overview** page. See [Tag Container Environments](#) for additional details.



Copy the Tag Loader to your Website

The loader that you can download on the **Environment Overview** page is used to asynchronously load the tag container. This file should be saved to your web server where it can be referenced by your site pages. Click the **Download JS Loader** button to save this file.

If you are an existing SiteCatalyst customer, you can save a copy of your current `s_code.js` file and replace it with the loader. If you are a new customer, save this file where it can be included by the pages on your site.

Include the Loader on Site Pages

Pages on your website use a script tag to include the Javascript loader. Variables and other properties are set on the page using the same process as with a standard deployment. You can reference the Implementation guide for your products for additional details.

We recommend including the page loader near the beginning of each page. The page code (and the call to `s.t()`) should be placed near the end of each page.

```
<script type="text/javascript" src="s_code.js"></script>
<script type="text/javascript">
if(s){
  s.pageName = ""
  s.server = ""
  s.channel = ""
  s.t()
}
//]]&gt;&lt;/script&gt;</pre>
</div>
<div data-bbox="83 307 572 324" data-label="Text">
<p>This sample code is available in the interface by clicking <b>View Page Code</b>.</p>
</div>
<div data-bbox="83 342 290 358" data-label="Section-Header">
<h2>Test and Promote Revisions</h2>
</div>
<div data-bbox="83 366 903 401" data-label="Text">
<p>When you are ready, use the links in the Actions column to copy a revision to the next environment. The current version in <b>Live Environment</b> is sent to users when they view a page that includes the tag loader.</p>
</div>
<div data-bbox="83 410 911 500" data-label="Table">
<table>
<tr>
<th data-cs="4" data-kind="parent">Live Environment</th><th data-kind="ghost"></th><th data-kind="ghost"></th><th data-kind="ghost"></th><th>File History</th></tr>
<tr>
<th>File Revision</th><th>Last Edited</th><th>Comments</th><th>Status</th><th>Actions</th></tr>
<tr>
<td>Revision 16</td><td>by bwatson on 09/16/11 at 10:09:03am</td><td>Demdex Added.</td><td>deployed on 09/16/11 at 10:17:03am</td><td><a href="#">Edit</a></td></tr>
</table>
</div>
<div data-bbox="83 521 340 541" data-label="Section-Header">
<h2>Migrating to a Tag Container</h2>
</div>
<div data-bbox="83 550 625 567" data-label="Text">
<p>A description of how to migrate your existing <code>s_code.js</code> file to a tag container.</p>
</div>
<div data-bbox="83 573 626 590" data-label="Text">
<p>Keep the following key concepts in mind as you create and deploy a tag container:</p>
</div>
<div data-bbox="83 596 920 709" data-label="List-Group">
<ul>
<li>• Tag containers are hosted on a CDN. Each time a new revision is deployed it takes several minutes to propagate to the edge nodes.</li>
<li>• Tag Containers are optimized to use asynchronous loading where possible to improve performance. If variables are not being populated or resources are not being found, make sure the code is inserted in the correct location.</li>
<li>• Each tag container has a dev, stage, and live environment. New tag containers are created with a dev version only. You must set a cookie to load the dev version. See <a href="#">Tag Container Environments</a>.</li>
<li>• Use the Preview button to display the generated tag container code to compare with your existing <code>s_code.js</code> file.</li>
</ul>
</div>
<div data-bbox="83 716 920 733" data-label="Text">
<p>The following table helps you identify the current code in your <code>s_code.js</code> file, and where it should be placed in a tag container:</p>
</div>
```

Description	Current s_code content	Tag Container Section	Action
Supporting Javascript	<pre>function ParsePageMetadata(name) { .. }</pre>	Custom Code	Place Javascript libraries and functions you use to populate variables in the Custom Code section. This makes them available before the s object is created.
SiteCatalyst	<pre>var s_account="rs" var s=s_gi(s_account) s.debugTracking=false s.charSet = "UTF-8" ... s.linkTrackEvents="None"</pre>	Product Code > SiteCatalyst	Add the SiteCatalyst product, and then overwrite the default code with your current code.
Plugins	<pre>s.usePlugins=true function s_doPlugins(s) { /* Add usage of plugins here */ } s.doPlugins=s_doPlugins</pre>	Product Code > SiteCatalyst	Overwrite the default plugins code that is generated in the SiteCatalyst Product Code section with your current plugins code.
Survey	<pre>s.loadModule("Survey")</pre>	Product Code > Survey	Add the Survey product, and then overwrite the default code with your current Survey module code.
Media Module	<pre>s.loadModule("Media")</pre>	Product Code > Video Tracking	Add the Video Tracking product, and then overwrite the default code with your current Media module code.
Genesis Integrations (Facebook, DFA, Mediamind)	<pre>s.Integrate.add("FBD");</pre>	Product Code > Genesis Integrations	<p>Add the integration and then copy your current integration code.</p> <p>If you have an s.Integrate section that does not match one of the listed integrations, insert a Dependent Code > JavaScript tag and insert your code.</p>

SiteCatalyst

When a new version of s_code is release, you are prompted to upgrade the next time you view the SiteCatalyst code block.

SiteCatalyst

Setting	Description
Accounts	Select one or more report suites to receive the data that is sent by this tag.
Secure Server	(Optional) If you have a secure server URL, provide it here. Leave it blank to default to the non-secure server URL.

Setting	Description
Character Set	Select the character set used by your report suite.
Currency Code	Select the currency code used by your report suite.
Instance Variable	If your instance variable name has changed from the default, provide the new variable name.

Marketing Cloud Deployment

Deploy Marketing Cloud products.

Marketing Cloud products are added by selecting the product in the **Add Product or Code** drop-down list when editing a tag container. The information required to add support for each product is listed in the following sections:

Survey

This section describes how to add the Survey module.

Integration with Survey is provided through the Survey module. After you add the module, you can define any additional settings required by your implementation. See [Standard Survey Implementations on the Web](#).

```
s.loadModule("Survey")
var s_sv_dynamic_root = "delivery.dl.sv.omtrdc.net/survey/dynamic"
var s_sv_gather_root = "collection.dl.sv.omtrdc.net/survey/gather"
```

AudienceManager

This section describes how to deploy AudienceManager tags.

TagManager connects to AudienceManager and retrieves the containers associated with your account. When you select a container, the code to support that container is added to your tag container.

Each time you make an update to the selected AudienceManager container, you need to open the AudienceManager product section in TagManager and click **Regenerate Core**.

Element	Description
Container ID	Select your assigned container ID.

JavaScript Video Measurement

This section describes how to add the Media module that is required to measure video using JavaScript.

Video measurement code is placed in the code block in the **Video Tracking** section.

Implementation details on measuring video are included in the [Measuring Video in SiteCatalyst](#) guide.

Managing Tags

Understand the tag container deployment process and add custom tags.

- Tag containers are hosted on a CDN. Each time a new revision is saved it takes several minutes to propagate to the edge nodes.
- Tag Containers are optimized to use asynchronous loading where possible to improve performance. If variables are not being populated or resources are not being found, make sure the code is inserted in the correct location.

- Each tag container has a dev, stage, and live environment. New tag containers are created with a dev version only. You must set a cookie to load the dev version. See [Tag Container Environments](#).

Add a Genesis Integration

TagManager generates the code required for common Genesis integrations.

Based on the integrations you use, you can add the following:

- Facebook
- DFA
- MediaMind

To add a Genesis Integration, select the integration in the **Product Code** section of your tag container. Configure the integration in the code block using the same process as with a standard implementation.

Add a Custom Integration

Custom integrations include all non-Genesis integrations that require read access to SiteCatalyst variables.

If your integration is not dependent on a Marketing Cloud product, it is often simpler to add it in the Custom Code (after products) section.

Custom integrations have access to the same Integrate module that is used by Genesis. The Integrate module provides methods to read and set variables on the instance object (`s` object). This lets you read data from another application to populate SiteCatalyst variables, and send SiteCatalyst variables (such as `s.pageName`) to another application.

Add a Dependent JavaScript Code Block

Custom integrations are added in a Dependent JavaScript Code block. To add a custom integration, add a **Dependent JavaScript Code** block in the **Product Code** section of your tag container.



Note: The name you provide for the code block is configured as the name of the integration in the code. When you create this tag, illegal characters and whitespace are removed. The name of the tag can be changed to a friendly name after the module is added.

Integrate module code is generated automatically similar to the following:

```
s.Integrate.add("module")
s.Integrate.module.setVars=function(s,p){
  /* Use this section to set additional partner data in SiteCatalyst variables */
  /* Example: s.campaign=p.cid */
};
s.Integrate.module.useVars=function(s,p){
  /* Use this section to send SiteCatalyst variables to your partner */
  /* Example: p.beacon("http://www.partner tracking service.com/track?page=[s.pageName]") */
};
```

Details on using the Integrate module are contained in the [SiteCatalyst Implementation Help](#).

Define Conditions

Custom integrations in Dependent Code blocks can be executed based on conditions. See [Conditional Execution](#).

TagManager Reference

Detailed topics that describe how TagManager works.

Custom Core JavaScript

This section is executed first and is where you define libraries, functions, and utilities that you want to use for all of your tags.

This code is executed before the instance variable (`s` object) is created. For example, if you have a custom function that parses metadata out of the page, you could declare it here.

```
function ParsePageMetadata(name)
{
  ...
}
```

To include a JavaScript library, you can copy and paste the code from the minified version directly into this section.

Product Code

This section contains implementation code for each product.

Each product that you define in this section is executed in the order listed. After product code is generated, you can modify the code that is inserted in the code block as you would with a standard implementation.

This biggest difference between using product code in TagManager and in a standard JavaScript file, is that TagManager executes product code asynchronously. This provides a number of performance benefits as your page continues to load while product code executes. Because of this, if you set a value for the same variable in a product section and on the page, there is no guarantee which value the variable contains. In this case you should set the variable in the `doPlugins` function as this code is executed after the code on the page.

Utility functions and libraries from the Custom Core JavaScript section can be used within any product section. For example, to use the function defined in the previous example, you could make a call similar to the following:

```
s.category = ParsePageMetadata(category);
```

DoPlugins

Code in this section is the last code executed before a hit is sent to SiteCatalyst.

You can place no plugin code here to avoid conflicts between JavaScript on the page and in the main product code section. You can add plugins to this section as you would with a standard implementation. This section is generated as part of the SiteCatalyst Product Code.

Dependent Code JavaScript

The Dependent Code section is executed after other product tags.

This section uses the Integrate module to read and write SiteCatalyst data. Tag Manager provides direct access to this module for custom integrations that need to use variable data from SiteCatalyst. If you have an integration that does not require SiteCatalyst data it can be added as a third-party tag or affiliate beacon.

Custom Code (after products)

Contains additional code to execute after product code.

Third-party tracking and affiliate beacons that do not rely on Marketing Cloud product data should be placed in this section. Tags that are placed in this section should function correctly whether or not you include any products.

Tags in the following formats are supported:

- JavaScript code block or remote JavaScript file
- Image beacon
- iFrame
- HTML block that contains one or more `` tags, `<script>` tags, or `<iframe>` tags. Other tags in this section are ignored.

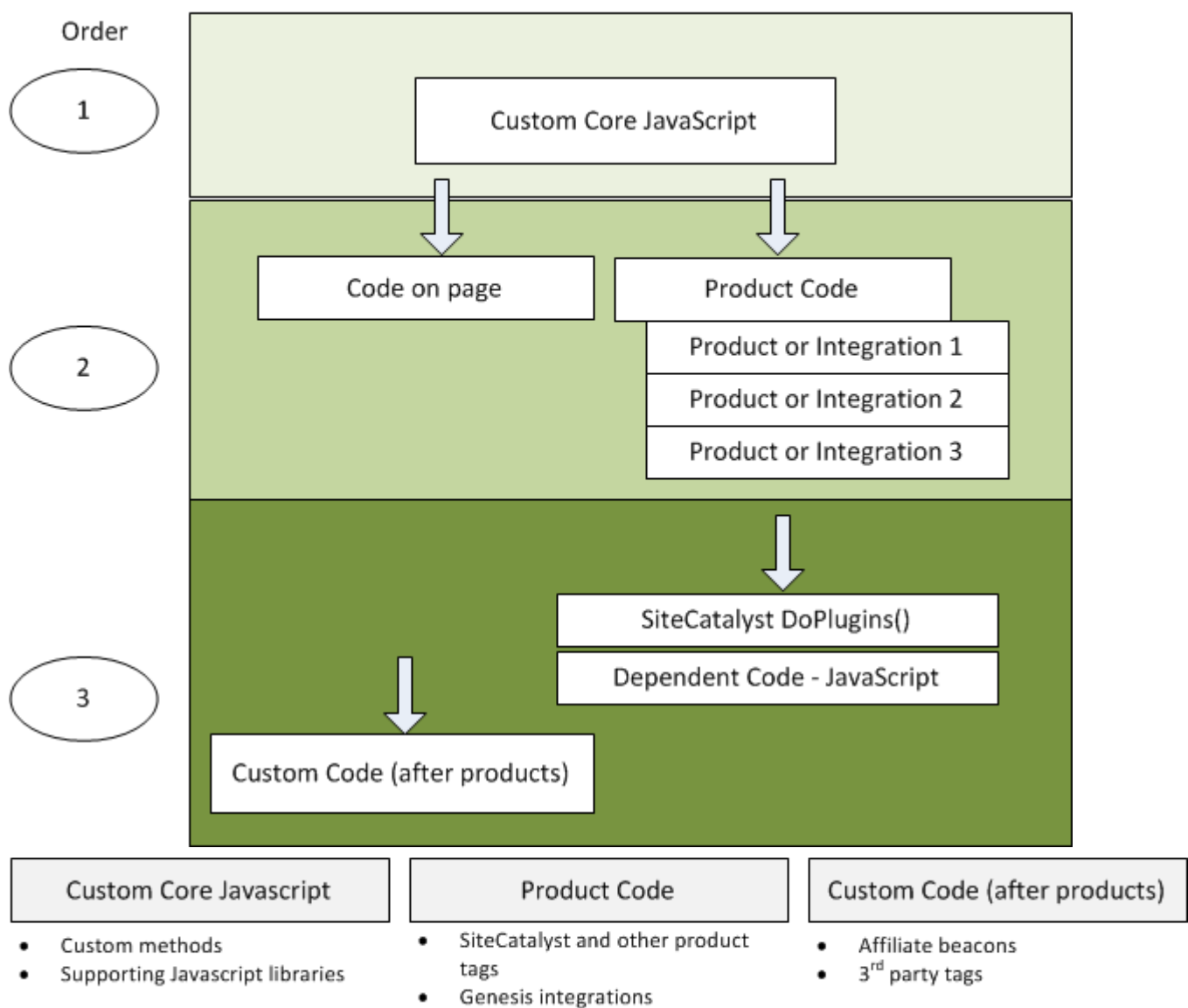
Code that is defined in the Custom Core JavaScript section is available in this section. Custom Code can be executed based on conditions. See [Conditional Execution](#).

Processing Order

The following figure contains a representation of asynchronous Tag Container processing.

Tag Container sections are executed in the order presented in the diagram. Each arrow should be considered as a separate thread, though the actual threads started during Tag Container processing vary according to your configuration.

Tag Container Processing Order



Conditional Execution

Tags defined in Dependent JavaScript Code Blocks and Custom Code (after products) blocks are fired conditionally based on matching criteria.

All conditions that you define for a code block must be met for the code to execute. For example, if you provide a Host Match and Path Match, both conditions must match.

Match strings are case-sensitive.

The following table contains a list of available conditions:

Block Component	Description
Date Range	Executes only within the defined date range.
Host Match	<p>Executes if the hostname contains the provided string.</p> <p>For the URL:</p> <p><code>http://en.main.example.co.uk/index.jsp?q=value</code></p> <p>The hostname is <code>en.main.example.co.uk</code></p>
Path Match	<p>Executes if any part of the path contains the provided string.</p> <p>For the URL:</p> <p><code>http://www.example.com/news/a.html?cid=ad1</code></p> <p>The path is <code>/news/a.html</code> .</p>
Query Parameter Match	<p>Executes if the defined query string parameter exists and the parameter value matches the provided string exactly.</p> <p>For the URL:</p> <p><code>http://www.example.com/a.html?cid=ad1&node=4</code></p> <p>The value of query string parameter <code>cid</code> is <code>ad1</code>, and the value of query string parameter <code>node</code> is <code>4</code>.</p> <p>Note that if the page URL is over 255 characters, query string parameters may be truncated.</p>
Cookie Match	Executes if the defined cookie exists and the cookie value matches the provided string exactly.

Advanced Conditional Matching

You can use JavaScript conditional statements in any code block to perform advanced conditional matching. For example, a JavaScript `if` statement that matches based on a regular expression can be inserted. If you want to send a call only on a specific page, you could add a JavaScript code block similar to the following:

```
<script>
var pageName = /MyPageURL.html/;
var pageURL = document.location.href;
if (pageURL.indexOf(pageName) != -1)
{
```

```
//send beacon  
}  
</script>
```

Permissions

All administrators have access to TagManager.

Using SiteCatalyst groups, you can define the following permissions for other users:

- Access to all tag containers in a specific environment (dev, stage, or live).
- Access to a specific tag container in all environments.

Assign Permissions

Admin > User Management > Groups> Click a group.

TagManager permissions can be assigned to any custom group or any individual members of a custom group.



Additional Implementation Resources

Adobe offers additional implementation resources, including the following white papers.

These white papers can be accessed via the Adobe Help site or the Adobe Knowledge Base.

- SiteCatalyst JavaScript Code - Upgrading from G Code to H Code
- SiteCatalyst JavaScript Code - Making an H Code JS File Compatible with G Page Code
- SiteCatalyst and Silverlight
- RightNow Technologies and SiteCatalyst
- Page Naming Strategies

Vulnerability Scanner

Whenever users are allowed to input values that are stored in SiteCatalyst or any other JavaScript variables, values should be properly escaped or URIEncoded.

We highly recommend using one of the free vulnerability scanning services, such as [Sitewatch](#), to avoid injection attacks and other potential exploits that might be present in your code.

SiteCatalyst Variables

SiteCatalyst uses several variables to store various values in memory. The values help perform various reporting functions in SiteCatalyst. For example, the value in the *pageName* variable is the name of the Web page being reported in SiteCatalyst.

Adobe uses plug-ins (which are programs that are added to your browser) by modifying the existing JavaScript code. This extends the capabilities of your browser to give you more functionality that is not available in the original application.

All of the variables and plug-ins that are created by Adobe are discussed below. Each variable and plug-in shows both syntax and other usage examples.

Variables and Limitations

The JavaScript file used to track Web pages is extremely versatile. However, potential problems and pitfalls may arise with an increased level of flexibility.

The following table lists the available SiteCatalyst variables:

Variable	Description
s_account	Determines the report suite where data is stored and reported in SiteCatalyst.
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.
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 SiteCatalyst visitor ID (s_vi) cookie will be set by determining the number of periods in the domain of the page URL. For www.mysite.com, <i>cookieDomainPeriods</i> should be "2." For www.mysite.co.jp, <i>cookieDomainPeriods</i> should be "3."
cookieLifetime	Used by both JavaScript and SiteCatalyst 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 SiteCatalyst databases. SiteCatalyst databases store all monetary amounts in a currency of your choice. If that

Variable	Description
	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 SiteCatalyst Conversion Module . When an eVar is set to a value for a visitor, SiteCatalyst 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 SiteCatalyst cookies other than the visitor ID (s_vi) 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.
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.

Variable	Description
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 JavaScriptActionSource ..
Media.trackVars	Identifies which variables should be sent with a media hit. It is only applicable with JavaScriptActionSource ..
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 ClickMap tracking or use ClickMap 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 SiteCatalyst.
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.
pageURL	In rare cases, the URL of the page is not the URL that you would like reported in SiteCatalyst. To accommodate these situations, SiteCatalyst 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.

Variable	Description
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 SiteCatalyst 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 SiteCatalyst. 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 ClickMap 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

Variable	Description
	which cookies are set. If <i>visitorNamespace</i> changes, all visitors reported in SiteCatalyst may become new visitors. In short, do not alter this variable without approval from an Adobe consultant.
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.

s_account

The *s_account* variable determines the report suite where data is stored and reported in SiteCatalyst.

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 SiteCatalyst 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 SiteCatalyst, 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 SiteCatalyst 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 SiteCatalyst.

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 [DigitalPulse Debugger](#).

```
http://mycompany.112.2o7.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 SiteCatalyst

The friendly name associated with each report suite ID can be changed by Adobe ClientCare. The friendly name can be seen in SiteCatalyst 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.

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** plugin, 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** plugin to force each campaign click-through to be counted only once per session. For more information on the **getValOnce** plugin, 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.

Examples

```
s.campaign="112233"
```

```
s.campaign=s.getQueryParam('cid');
```

SiteCatalyst 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 SiteCatalyst Admin Console.

Pitfalls, Questions, and Tips

- To keep click-throughs from being inflated, use the **getValOnce** plugin 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 in SiteCatalyst, see [Campaigns](#) in the SiteCatalyst User Guide.
- Use the DigitalPulse Debugger to see the actual value of campaigns (v0 in the debugger) sent into SiteCatalyst. If v0 does not appear in the debugger, no campaign data is recorded for that page.

charSet

SiteCatalyst uses the *charSet* variable to translate the character set of the Web page into UTF-8.

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	" "

The *charSet* variable is used to identify the character set of the page. For more information on character sets, see the [Multi-byte Character Sets](#) white paper before using the *charSet* variable.

Syntax and Possible Values

The *charSet* variable may only contain one of a predefined set of values, as listed in [Multi-byte Character Sets](#).

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

Examples

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

```
s.charset="SJIS"
```

Pitfalls, Questions, and Tips

- The value of *charset* must match the possible values listed in [Multi-byte Character Sets](#).
- The value of *charset* should reflect the character set of the 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. For more information, see the Channels and Hierarchies white paper.

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

cookieDomainPeriods

The *cookieDomainPeriods* variable determines the domain on which the SiteCatalyst 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, ClickMap 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 [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"

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 ClickMap 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.

cookieLifetime

The *cookieLifetime* variable is used by both JavaScript and SiteCatalyst 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 SiteCatalyst servers, with one exception, described below. The *cookieLifetime* variable can have one of three values:

- SiteCatalyst 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
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

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

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

currencyCode

The *currencyCode* variable determines the conversion rate to be applied to revenue as it enters the SiteCatalyst databases.

SiteCatalyst databases store all monetary amounts 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 SiteCatalyst 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 recommended 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 SiteCatalyst.

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 treated in SiteCatalyst, 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"
```

SiteCatalyst Configuration Settings

Adobe ClientCare 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 SiteCatalyst 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 in the SiteCatalyst Admin Console.
- When the `currencyCode` variable is empty, no conversion is applied.

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**.

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 ClickMap 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;
```

SiteCatalyst 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 SiteCatalyst 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.

dynamicAccountList

The SiteCatalyst JavaScript file 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>mysuite2=www2.mycompany.com;mysuite1=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"
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

- 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*.

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	window.location.host

Syntax and Possible Values

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

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

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

Examples

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

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

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

- 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.

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
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

Always use the DigitalPulse Debugger to determine which report suite is receiving data from each page.

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.

eVarN

The **eVar** variables are used for building custom reports within the SiteCatalyst **Conversion Module**.

When an eVar is set to a value for a visitor, SiteCatalyst 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.

Max Size	Debugger Parameter	Reports Populated	Default Value
255 Bytes	V1-v75	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 SiteCatalyst to 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," SiteCatalyst 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 SiteCatalyst 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).

Visits and Visitors

In order to see the number of **Visits** or **Daily Unique Visitors** associated with a conversion variable, Adobe ClientCare must enable it in SiteCatalyst.

Syntax and Possible Values

While eVars may be renamed within SiteCatalyst, they should always be referred to in the JavaScript file by eVarX, where X is a number between 1 and 75.

```
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"
```

SiteCatalyst Configuration Settings

eVars can be configured in the SiteCatalyst 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	<p>Allows you to change the name of the eVar report within SiteCatalyst.</p> <p>The eVar should still be referenced as s.eVarX in the JavaScript code, no matter what name is given to the report in SiteCatalyst.</p>
Type	<p>Allows you to show whether the eVar is a Text String or Counter.</p>
Allocation	<p>Used to configure which value of the eVar receives credit for success events.</p> <p>If Allocation is set to "Most Recent (Last)," then B receives credit.</p> <p>If Allocation is set to "Original Value (First)" then A receives credit.</p> <p>If Allocation is set to "Linear", then both A and B receive credit for half the purchase value.</p>
Expire After	<p>Lets you determine whether an eVar expires on a specific event, like purchase, or after a custom or predefined time period.</p>
Reset	<p>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.</p>

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 SiteCatalyst reports.
- eVar counters may not contain negative numbers.

events

The *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 SiteCatalyst 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 SiteCatalyst implementations. The seven pre-configured conversion events are automatically enabled for all new report suites. New custom events (event1-event100) can be enabled by any admin-level user in SiteCatalyst by using the SiteCatalyst 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 - event100	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="shoes;SKU1234;1;99.99;event1=4.50"
```

```
s.events="event1,event4"
```

```
s.products="shoes;SKU1234;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=";Shoes;1;69.95;;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 in SiteCatalyst 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"
```

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 SiteCatalyst 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 SiteCatalyst 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 ClickMap 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"
```

SiteCatalyst Configuration Settings

None

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 ClientCare. 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 ClientCare 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"
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

- Before implementing hierarchies, see the Channels and Hierarchies white paper.
- The delimiter may not be changed after the hierarchy is set up. If the delimiter for your hierarchy must be changed, contact Adobe ClientCare.
- 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

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](#)
- [SiteCatalyst 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 to SiteCatalyst 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"
```

SiteCatalyst 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.

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=""
```

SiteCatalyst 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".

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	"javascript:"

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". 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 to SiteCatalyst. If you would like multiple domains or companies to use a single SiteCatalyst 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="javascript:,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="javascript:,site1.com[,site2.com[,site3.net[...]]]"
```

The default value of *linkInternalFilters* contains "javascript:" because any links to a JavaScript function are not usually exit links. If "javascript:" were not in the list, then all links to JavaScript functions would be considered external.

Examples

```
s.linkInternalFilters="javascript:,mysite.com"
```

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

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

- Include all domains that the SiteCatalyst 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="javascript:,mysite.com,mysite.net,mypartner.net/adclick"
```

linkLeaveQueryString

By default, query strings are excluded from all SiteCatalyst 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 in SiteCatalyst

No SiteCatalyst 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, ClickMap, or **Path** reports.

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 Exit Links	""

Custom Links refer to links that send data to SiteCatalyst. 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"
```

SiteCatalyst 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).

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 SiteCatalyst, 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"
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

- The JavaScript file only uses *linkTrackEvents* if *linkTrackVars* contains the "events" variable.
- 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.

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 SiteCatalyst, 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"
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

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

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 SiteCatalyst. 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 may only contain a single character, namely 'o', 'e', or 'd.'

```
s.tl(this, 'o', 'Link Name');
```

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>
```

SiteCatalyst 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.
- Page Views are the only metric available by default on SiteCatalyst 14. *Participation metrics* are available if requested.
- SiteCatalyst 15 introduces visits and unique visitors to almost every report in SiteCatalyst, 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 not deduplicated.

To change a prop into a list prop, a Supported User must contact ClientCare and provide:

- The Report Suite ID (RSID).

- The traffic variable you wish to become a list prop.
- The list prop delimiter. Popular ones are colons, semi-colons, commas, or pipes. It can technically be any of the first 127 ASCII characters.

List props are not visible in the Admin Console. As such, take note of this setting in either the prop's report or outside of SiteCatalyst.

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 are available only on the SiteCatalyst 15 processing platform.
- List Vars remember their specific values by referencing the VisitorID cookie in the visitor's browser.
- Each delimited value can contain a maximum of 255 characters (or less if using multi-byte characters).
- 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 cannot use SAINT classifications.
- 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

An Adobe representative must enable List Vars. The following customizations are available, which are to be included when requesting to enable one:

- **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.
- **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 unique values each image request is allowed. This setting is optional.

To begin using List Vars or enable another List Var, please have one of your organization's supported users contact ClientCare with the following information:

- The delimiter you want to use to separate each value
- When you want the variable to expire
- Whether to use full or linear allocation
- (Optional) The maximum number of values that are allowed per image request

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";
```

An example 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.

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 SiteCatalyst hit data.

Losing SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst 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
100 Bytes	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 SiteCatalyst 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`.

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 SiteCatalyst 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.

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="" //cookies used first, produces /5/ 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 SiteCatalyst 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 SiteCatalyst variables (ASCII characters with codes above 127), SiteCatalyst 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 in SiteCatalyst. 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 SiteCatalyst.

```
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"
```

SiteCatalyst Configuration Settings

Administrators have the ability to change the visible page name in SiteCatalyst with the **Name Pages** tool, which is potentially dangerous and may negatively affect your reports. Please contact Adobe ClientCare 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"
```

SiteCatalyst 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 SiteCatalyst.

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 SiteCatalyst image requests. To prevent truncation of other data, the first 256 bytes of the *pageURL* are specified early in the request string, with all characters after the first 256 bytes appended to the end of the request.

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 SiteCatalyst reports, and the query string may be stripped, depending on SiteCatalyst 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 ClientCare before using the *pageURL* variable for custom purposes.

Examples

```
s.pageURL="http://mysite.com/home.jsp?id=1224"
```

```
s.pageURL="http://www.mysite.com/"
```

SiteCatalyst 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.

The *products* variable should always be set in conjunction with a success event. The *products* variable can track numeric and currency events, as well as **Merchandising** eVars.

Max Size	Debugger Parameter	Reports Populated	Default Value
No Limit*	products	Products Categories (optional) Revenue (optional) Units (optional) Custom Events (optional) eVars (optional)	" "



Note: Although Adobe does not impose a size limit on products, most browsers impose a size limit on the URL of SiteCatalyst image requests. The **"Product"** and **"Category"** sections of products each have a limit of 100 bytes.

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.

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="Running;Shoe"
```

Example 2: Multiple Products

```
s.events="prodView"
s.products="Running;Shoe,Running;Socks"
```

Example 3: Omitting Category

```
s.events="prodView"
s.products=";Shoe,;Socks"
```



Note: The category/product delimiter (;) is required as a place holder when omitting categories.

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="Running;Shoe;1;69.95"
s.purchaseID="1234567890"
```

Example 2: Multiple Products

```
s.events="purchase"
s.products="Running;Shoe;1;69.95,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 "Category;Widget;3;59.97").

Example 3: Omitting Category

```
s.events="purchase"
s.products=";Shoe;1;69.95,;Socks;10;29.99"
s.purchaseID="1234567890"
```



Note: The category/product delimiter (;) is required as a place holder when omitting categories.

Example 4: Multiple Products with an Additional Currency Event

```
s.events="purchase;event10=7.95"
s.products=";Shoes;1;69.95,;Socks;10;29.99"
s.purchaseID="1234567890"
```

Numeric/Currency Events that are given a value in the events list apply to all products in the products list.



Note: The category/product delimiter (;) is required as a place holder when omitting categories.

Setting products with Numeric and Currency Events

By default, success events are configured as "counter" events. Counter events simply count the number of times a success event is set. Some success event uses require that an event be incremented by some custom amount. These events can be set either as "numeric" events or "currency" events. See [events](#).

Syntax

```
s.events="event1"
s.products="Category;Product;Quantity;Price;eventN=X[|eventN=X][,Category;Product;Quantity;Price;eventN=X]"
```

Example 1: Single Numeric/Currency Event

```
s.events="purchase,event1"
s.products="Running;Shoe;1;69.95;event1=7.59"
```

Example 2: Multiple Numeric/Currency Events

```
s.events="purchase,event1,event2"
s.products="Running;Shoe;1;69.95;event1=7.59|event2=19.45"
```

Setting products with Merchandising eVars

See the [Cross-Category Merchandising](#) white paper.

Configuring the Variable in SiteCatalyst

Configuration for events, including setting them as **counter**, **numeric**, or **currency** events, can be done in the SiteCatalyst Admin Console by going to **Edit Settings > Conversion > Success Events**. Event types should never be changed after the event begins collecting data.

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.
- The category represents the "Home" category for the product. The product-category relationship is created when a product is first recorded and persists indefinitely. All subsequent success events recorded for the product are automatically credited to the product's "Home" category.

propN

Property (**prop**) variables are used for building custom reports within the SiteCatalyst **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"
```

SiteCatalyst Configuration Settings

Contact Adobe ClientCare about showing **Visit**, **Visitor**, and **Path** metrics for **prop** variables.

purchaseID

The *purchaseID* is used to keep an order from being counted multiple times by SiteCatalyst.

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"
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

The *purchaseID* variable allows all conversion variables on the page to be counted only once in SiteCatalyst 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 SiteCatalyst 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')
```

SiteCatalyst 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 ClickMap tracking or use ClickMap to report on a link type or location, rather than the link URL.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 Bytes	OID	ClickMap	The Absolute URL of a Clicked Link

There are three common reasons to use *s_objectID*:

- To aggregate clicks on links that change often during a day
- To separate clicks on links that ClickMap combines
- To improve accuracy of ClickMap 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 ClickMap. 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, ClickMap reports on those links at an individual product basis.

Improving ClickMap 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. Using *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"
```

SiteCatalyst 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 **SiteCatalyst** reports. The SiteCatalyst 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
```

SiteCatalyst 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 > States	""

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"
```

SiteCatalyst 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.

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 timezone can be specified in standard or extended format (\pm HH, \pm HH:MM, \pm HH, \pm 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
```

SiteCatalyst Configuration Settings

A report suite must be enabled to accept custom timestamps by ClientCare 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.

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 SiteCatalyst. The data that is sent with a download link includes the link download URL, and ClickMap data for that link. If *trackDownloadLinks* is 'false,' then ClickMap 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
```

SiteCatalyst 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 ClickMap.
- When *trackDownloadLinks* is 'true,' data is sent to SiteCatalyst each time a visitor clicks a file download link.

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, data is immediately sent to SiteCatalyst. The data that is sent with an exit link includes the link URL, link name, and ClickMap data for that link. If *trackExternalLinks* is 'false,' then ClickMap 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
```

SiteCatalyst 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 ClickMap.
- When *trackExternalLinks* is 'true,' data is sent to SiteCatalyst each time a visitor clicks on an exit link (before link target loads).

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	""

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	""

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	True

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
```

SiteCatalyst Configuration Settings

None

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.

Max Size	Debugger Parameter	Reports Populated	Default Value
100 bytes	xact	n/a	""

Syntax and Possible Values

```
s.transactionID="unique_id"
```

The *transactionID* should not contain a pipe character.

Examples

```
s.transactionIS-"11123456"
```

```
s.transactionID="lead_12345xyz"
```

```
s.transactionID=s.purchaseID
```

SiteCatalyst Configuration Settings

Before *transactionID* values are recorded, *transactionID* recording must be enabled by an Adobe representative. To see whether *transactionID* recording is enabled for a report suite, go to SiteCatalyst > **Data Sources** > **Manage**.

Pitfalls, Questions, and Tips

- If multiple **transactionIDs** should be recorded in a single hit, you can use a pipe character to delimit multiple values. 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 an Adobe representative 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*.

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.

SiteCatalyst Configuration Settings

None

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.

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"
```

SiteCatalyst 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 SiteCatalyst 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	N/A	N/A	""

SiteCatalyst 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' SiteCatalyst 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"
```

Examples

```
s.visitorNamespace="company_name"
```

```
s.visitorNamespace="Adobe"
```

SiteCatalyst Configuration Settings

None

Pitfalls, Questions, and Tips

- If you use a CNAME to make SiteCatalyst cookies into first-party cookies, the **s.visitorNamespace** variable may be left blank. It is acceptable to leave it as constituted.
- If your third-party cookie implementation uses the **s.trackingServer** variable, you may not have an **s.visitorNamespace** variable. This is normal. The **s.trackingServer** variable serves the same purpose as **s.visitorNamespace**.

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"
```

SiteCatalyst 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.

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 in SiteCatalyst using Processing Rules. Processing Rules provides 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.

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	<p>Context data variable names can contain only alphanumeric characters, underscores and dots. Any additional characters are stripped out. ContextData variables do not have a numeric designation. Rather, they are named.</p> <p>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>.</p>
Namespace	<p>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.</p> <p>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.</p>
URL Limits	<p>When using Context Data variables on the Web, be aware of the URL limits of browsers. For example, IE truncates URLs at 2000 bytes. You can use the DigitalPulse debugger to determine the size of a URL string.</p>
Supported AppMeasurement version	<p>ContextData variables require at least H23 code or higher.</p>

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.

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"
```

Plug-ins

A plug-in is an add-on for a program that adds functionality. Marketing Cloud JavaScript plug-ins can greatly enhance the core functionality of the SiteCatalyst code.

You can use these plug-ins to perform any of the following functions.

- Capture internal search keyword and marketing campaign data
- Prevent click-throughs and eVar instances from being counted on page reloads
- Return the value of a specified query string parameter, if found in the current page URL
- Append a value to any delimited lists
- Force a variable to be populated only once within a single session or time period

In addition to these plug-ins, the functionality of your implementation can be extended by working with ClientCare, who can analyze your key business requirements and provide customized solutions to help you leverage SiteCatalyst and other Marketing Cloud products.

Popular plug-ins are provided here. However, this is not a comprehensive list of plug-ins. Contact ClientCare for additional help with plug-ins

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.
---------	--

Implementation

Follow these steps to implement the APL plug-in.

- 1. Request the plug-in code from ClientCare 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	<code>apl((L,v,d,u)</code> L= source list, empty list is accepted v = value to append d = list delimiter 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.
Return Value	original list, with appended value if added
Usage Examples	<code>s.events=s.apl(s.events,"event1"," ",1);</code>

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.toLowerCas"
+"e()));}}if(!m)l=1?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

```

Cookie Combining Utility

Utility that reduces the number of cookies set by Adobe code. Data for all cookies are stored within one session cookie and one persistent cookie.

detectRIA

A plug-in that determines whether users on your site have Adobe Flash or Microsoft Silverlight installed in their browsers. If so, the plug-in sets the version numbers in SiteCatalyst variables.

This plug-in is useful to determine the versions of these applications that should be used in developing rich media content on your site. For example, if 46% of site users only have Flash 9 installed (when the current version is Flash 10), you may consider developing for Flash 9 rather than requiring Flash 10 to use your applications.



Note: The following instructions require you to alter the SiteCatalyst JavaScript 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 SiteCatalyst.

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 capturing RIA

data. These should be variables that have been enabled using the Admin Console, but which are 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.detectRIA('s_ria','prop1','prop2','10','2','');
```

s.detectRIA has six arguments:

1. Name of the cookie that stores values for this plug-in (*s_ria* above).
2. Variable for use in capturing Flash version information (*prop1* above).
3. Variable for use in capturing Silverlight version information (*prop2* above).
4. Max Flash version to detect (default, if left blank, is 10; applies to IE only).
5. Max Silverlight version to detect (default, if left blank, is 2; applies to IE only).
6. Flag for detection once per visit. If left blank, the plug-in captures Flash version on every page view. If set to any value, detection occurs only on the first page view of the visit.

Other Examples

```
/* Does not set version max and uses eVars instead of props */
s.detectRIA('s_ria','eVar6','eVar7','','','');
```

```
/* Sets flag to capture version numbers once per visit*/
s.detectRIA('s_ria','prop1','prop2','','','1');
```

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: detectRIA v0.1 - detect and set Flash, Silverlight versions
 */
s.detectRIA=new Function("cn", "fp", "sp", "mfv", "msv", "sf", ""
+"cn=cn?cn:'s_ria';msv=msv?msv:2;mfv=mfv?mfv:10;var s=this,sv='',fv=-"
+"1,dwi=0,fr='',sr='',w,mt=s.n.mimeTypes,uk=s.c_r(cn),k=s.c_w('s_cc',"
+"true',0)?'Y':'N';fk=uk.substring(0,uk.indexOf('|'));sk=uk.substrin"
+"g(uk.indexOf('|')+1,uk.length);if(k=='Y'&&s.p_fo('detectRIA')){if(u"
+"k&&!sf){if(fp){s[fp]=fk;}if(sp){s[sp]=sk;}return false;}if(!fk&&fp)"
+"if(s.pl&&s.pl.length){if(s.pl['Shockwave Flash 2.0'])fv=2;x=s.pl["
+"Shockwave Flash'];if(x){fv=0;z=x.description;if(z)fv=z.substring(16"
+"z.indexOf('.')}else if(navigator.plugins&&navigator.plugins.len"
+"gth){x=navigator.plugins['Shockwave Flash'];if(x){fv=0;z=x.descript"
+"ion;if(z)fv=z.substring(16,z.indexOf('.')}else if(mt&&mt.length)"
+"{x=mt['application/x-shockwave-flash'];if(x&&x.enabledPlugin)fv=0;}"
+"if(fv<=0)dwi=1;w=s.u.indexOf('Win')!=-1?1:0;if(dwi&&s.isie&&w&&exec"
+"Script){result=false;for(var i=mfv;i>=3&&result!=true;i--){execScri"
+"pt('on error resume next: result = IsObject(CreateObject(\"Shockwav"
+"eFlash.ShockwaveFlash.'+i+'\"))','VBScript');fv=i;}}fr=fv==1?'flas"
+"h not detected':fv==0?'flash enabled (no version)':'flash '+fv;if("
+"!sk&&sp&&s.apv>=4.1){var tc='try{x=new ActiveXObject(\"AgControl.A"
+"gControl\");for(var i=msv;i>0;i--){for(var j=9;j>=0;j--){if(x.is"
+"VersionSupported(i+\".\")+j)}{sv=i+\".\")+j;break;}}if(sv){break;}"
+"+\"}}catch(e){try{x=navigator.plugins[\"Silverlight Plug-In\"];sv=x"
+"+\".description.substring(0,x.description.indexOf(\".\")+2);}catch("
+"+\"e){}}';eval(tc);sr=sv=='?'silverlight not detected':'silverlight"
+" '+sv;}if((fr&&fp)|| (sr&&sp)){s.c_w(cn,fr+'|'+sr,0);if(fr)s[fp]=fr;"
+"if(sr)s[sp]=sr;}}");
s.p_fo=new Function("n","",
+"var s=this;if(!s.__fo){s.__fo=new Object;if(!s.__fo[n]){s.__fo[n]="
+"new Object;return 1;}else {return 0;}}");
```

Always test plug-in installations extensively to ensure that data collection is as expected before deploying in a production environment.

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 SiteCatalyst 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.1"
//
// ... more code below ...
//
```

getAndPersistValue

The getAndPersistValue plug-in obtains a value of your choosing and populates it into a SiteCatalyst 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.

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 SiteCatalyst JavaScript 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 SiteCatalyst.

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 SiteCatalyst variable.

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 in DataWarehouse, Discover, or ASI. 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 SiteCatalyst JavaScript code on your site. Altering code can affect data collection and should be done only by a developer with experience using and implementing SiteCatalyst.

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*.

getNewRepeat

Determines whether a visitor is a new visitor or a repeat visitor, and captures this information in a SiteCatalyst 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 SiteCatalyst JavaScript 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 SiteCatalyst.

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.

getPercentPageViewed

Record the portion of a page (0-100%) that the user views and pass the value into a variable on the next page view. This plugin lets you determine how much of your content users are seeing on average, so that you can optimize your page lengths and layouts based on user behaviors.

You can also use SAINT to group percentages (such as *Less than 25%* or *Less than 50%*) so that you can identify page-viewing trends.



Note: The following instructions require you to alter the SiteCatalyst JavaScript 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 SiteCatalyst.

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.getPercentPageViewed();
```

Returns

- array[0] contains the previous page identifier
- array[1] contains the total percent viewed
- array[2] contains the percent viewed on initial load
- array[3] contains the total number of vertical pixels viewed

Sample Calls

```
/* Plugin Example: getPercentPageView v1.4 total percent viewed only*/
s.prop2 = s.getPercentPageViewed();

/* Plugin Example: getPercentPageView v1.4 multiple values*/
var ppvArray = s.getPercentPageViewed(s.pageName);
s.prop1 = ppvArray[0] //contains the previous page name
s.prop2 = ppvArray[1] //contains the total percent viewed
s.prop3 = ppvArray[2] //contains the percent viewed on initial load
s.prop4 = ppvArray[3] //contains the total number of vertical pixels

/* To send the hit on an exit link multiple values*/
var url = s.exitLinkHandler()
if(url) {
    s.linkTrackVars = 'prop1,prop2,prop3,prop4';
    var ppvArray = s.getPercentPageViewed(s.pageName);
    s.prop1 = ppvArray[0]
    s.prop2 = ppvArray[1]
    s.prop3 = ppvArray[2]
    s.prop4 = ppvArray[3]
}
```

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: getPercentPageViewed v1.4
```



```

*/
s.handlePPVEvents=new Function("", ""
+"if(!s.getPPVid)return;var dh=Math.max(Math.max(s.d.body.scrollHeight"
+"t,s.d.documentElement.scrollHeight),Math.max(s.d.body.offsetHeight,"
+"s.d.documentElement.offsetHeight),Math.max(s.d.body.clientHeight,s."
+"d.documentElement.clientHeight)),vph=s.wd.innerHeight||(s.d.documen"
+"tElement.clientHeight||s.d.body.clientHeight),st=s.wd.pageYOffset||"
+"(s.wd.document.documentElement.scrollTop||s.wd.document.body.scroll"
+"Top),vh=st+vph,pv=Math.min(Math.round(vh/dh*100),100),c=s.c_r('s_pp"
+"v'),a=(c.indexOf(',')>-1)?c.split(',')>4?[]:id=(a.length>0)?(a[0]):"
+"escape(s.getPPVid),cv=(a.length>1)?parseInt(a[1]):(0),p0=(a.length>"
+"2)?parseInt(a[2]):(pv),cy=(a.length>3)?parseInt(a[3]):(0),cn=(pv>0)"
+"?(id+','+((pv>cv)?pv:cv)+','+p0+','+((vh>cy)?vh:cy)):'';s.c_w('s_pp"
+"v',cn);");
s.getPercentPageViewed=new Function("pid",""
+"pid=pid?pid:'-';var s=this,ist=!s.getPPVid?true:false;if(typeof(s.l"
+"inkType)!='undefined'&&s.linkType!='e')return'';var v=s.c_r('s_ppv'"
+""),a=(v.indexOf(',')>-1)?v.split(',')>4?[]:if(a.length<4){for(var i="
+"3;i>0;i--){a[i]=(i<a.length)?(a[i-1]):('');}a[0]='';a[0]=unescape("
+"a[0]);s.getPPVpid=pid;s.c_w('s_ppv',escape(pid));if(ist){s.getPPVid"
+""=(pid)?(pid):(s.pageName?s.pageName:document.location.href);s.c_w('s"
+"s_ppv',escape(s.getPPVid));if(s.wd.addEventListener){s.wd.addEventL"
+"istener('load',s.handlePPVEvents,false);s.wd.addEventListener('scro"
+"ll',s.handlePPVEvents,false);s.wd.addEventListener('resize',s.handl"
+"ePPVEvents,false);}else if(s.wd.attachEvent){s.wd.attachEvent('onlo"
+"ad',s.handlePPVEvents);s.wd.attachEvent('onscroll',s.handlePPVEvent"
+"s);s.wd.attachEvent('onresize',s.handlePPVEvents);}}return(pid!='-'?"
+"")?(a):(a[1]);");

```

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 percentage of page viewed on the previous page, data is not collected for the final page view of the visit. One way around this is to use a function on all exit links to call *s.getPercentPageViewed()*.
- 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 SiteCatalyst.
- The plug-in creates its own first-party cookie named *s_ppv*.
- A very small percentage of users will not pass percentage of page viewed data due to browser limitations. However, this plug-in has been successfully tested in IE, Firefox, Chrome, Safari, and Opera.

getPreviousValue

Captures the value of a SiteCatalyst 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 SiteCatalyst JavaScript 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 SiteCatalyst.

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,'gpv_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 SiteCatalyst variables.

Once installed in your SiteCatalyst JavaScript code, the plug-in is configured by selecting a SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst 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.

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.location)
+"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 SiteCatalyst reports with hour of day, day of week, and weekend and weekday values.

This plug-in is useful in performing analysis based on the time of day or day of week. It can help to answer the following questions:

- Across a large date range, what is the most popular time of day for visitors to access my site?
- Are there days of the week on which conversion is higher on my site?
- How do my weekend sales compare to my weekday sales?
- Does a certain marketing campaign generate higher conversions in the morning, or in the afternoon?

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 SiteCatalyst JavaScript code. Altering the code can affect data collection and should be performed by a developer with experience using and implementing SiteCatalyst.

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.

```
// timeparting config variables

s.dstStart="1/1/2008"; // update to the correct Daylight Savings Time start date for the current
year.
s.dstEnd="1/1/2008"; // update to the correct Daylight Savings Time end date for the current
year.
s.currentYear="2008"; // update to the current year (can be done programmatically).
```

Plug-in Config

Place the following code in the `s_doPlugins()` function, which is located in the `s_code.js` file in the **Plugin Config** section. You can choose three Custom Traffic (`s.prop`) variables and three Custom Conversion (`s.eVar`) variables to capture time-parting data. These variables should have been enabled in the Admin Console but are not currently in use. The following is given as an example, and should be updated appropriately based on your needs.

```
s.prop1=s.getTimeParting('h','-7'); // Set hour
s.prop2=s.getTimeParting('d','-7'); // Set day
s.prop3=s.getTimeParting('w','-7'); // Set weekday
```

This plug-in has two arguments:

1. Type of data: h = hour of day, d = day of week, w = weekday/weekend
2. Report suite time zone: Enter the GMT offset for the destination report suite. For example, if your report suite uses U.S. Pacific Time, enter -8 for the second argument. If Eastern European Time, enter +2.

Other Examples

```
/* Uses GMT time zone and eVars instead of props */
s.eVar7=s.getTimeParting('h','0'); // Set hour
s.eVar8=s.getTimeParting('d','0'); // Set day
s.eVar9=s.getTimeParting('w','0'); // Set weekday
```

```
/* Uses GMT +2 time zone */
s.eVar7=s.getTimeParting('h','+2'); // Set hour
s.eVar8=s.getTimeParting('d','+2'); // Set day
s.eVar9=s.getTimeParting('w','+2'); // Set weekday
```

PLUGINS SECTION

Add the following code to the **PLUGINS SECTION** in the `s_code.js` file.

```
/*
 * 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}};");
```

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.
- Format for config variables is MM/DD/YYYY.
- s.dstStart is the start day, month and year for Daylight Savings Time.
- s.dstEnd is the end day, month and year for Daylight Savings Time.
- s.CurrentYear is the current year. If not set, will assume year of browser.
- If country has no DST, set dstStart, dstEnd to default (1/1/2009).
- If DST crosses into new year, set dstStart to month, day, year of previous year.

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.

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 SiteCatalyst 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 at 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 SiteCatalyst defines as a session (or visit). SiteCatalyst 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 SiteCatalyst visit open.
 - If a user keeps the browser window open without clicking on a link for more than 30 minutes, the SiteCatalyst visit expires while keeping the browser session open.

Implementation

To implement this plug-in, place the following code within your `s_code.js` file

```
/*
 * Plugin: getValOnce_v1.1
 */
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');
```

getVisitNum

The *getVisitNum* plug-in determines how many visits a user has made to your site and captures this number in a SiteCatalyst 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();
```

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: Visit Number By Month 2.0 - Return the user visit number
 */
s.getVisitNum=new Function("
+ \"var s=this,e=new Date(),cval,cvisit,ct=e.getTime(),c='s_vnum',c2='s\"
+ \"_invisit';e.setTime(ct+30*24*60*60*1000);cval=s.c_r(c);if(cval){var\"
+ \" i=cval.indexOf('&vn='),str=cval.substring(i+4,cval.length),k;}cvis\"
+ \"it=s.c_r(c2);if(cvisit){if(str){e.setTime(ct+30*60*1000);s.c_w(c2,\"
+ \"true',e);return str;}else return 'unknown visit number';}else{if(st\"
+ \"r){str++;k=cval.substring(0,i);e.setTime(k);s.c_w(c,k+'&vn='+str,e)\"
+ \"';e.setTime(ct+30*60*1000);s.c_w(c2,'true',e);return str;}else{s.c_w\"
+ \"(c,ct+30*24*60*60*1000+'&vn=1',e);e.setTime(ct+30*60*1000);s.c_w(c2\"
+ \"', 'true',e);return 1;}}\"
+ \" );
```

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.

manageQueryParam

Manages query string parameters by either encoding, swapping, or both encoding and swapping a value. For SearchCenter use only.

trackTNT

Collects the information for the Test&Target to SiteCatalyst integration.

The s.t() Function

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 SiteCatalyst. 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>
```

Based on SiteCatalyst **NetAverages**, approximately 5-10% of users do not support Javascript. The **<noscript>** tag fires a hardcoded image request for these users so they can be tracked. Internet Explorer however, executes the **<noscript>** tag regardless of whether Javascript is disabled or not, which is why the second line of code is present. If it recognizes Internet Explorer as the browser, it skips the **<noscript>** tag, which prevents double counting.

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.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 SiteCatalyst 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();
```

Link Tracking

SiteCatalyst automatically tracks file downloads and exit links based on parameters set in the 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.

Type	Report Location
File Downloads	Site Content > Links > File Downloads
Exit Links	Site Content > Links > Exit Links
Custom Links	Site Content > Links > Custom Links

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 <a href='http://othersite.com/index.html?r=mysite.com
```

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'>
```



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.

Type	Description and Report Location
File Downloads	Any URL with a file type that is listed in <code>linkDownloadFileTypes</code> is considered a download link. Site Content > Links > File Downloads
Exit Links	Any URL that does not contain one of the values in <code>linkInternalFilters</code> is considered an exit link. Determination of an exit link is modified by the setting of <code>linkLeaveQueryString</code> . Site Content > Links > 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 a SiteCatalyst 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 three arguments:

```
s.tl(this,linkType,linkName, variableOverrides, doneAction)
```

Custom Link Tracking on WebKit Browsers

JavaScript H.25 includes an update to ensure that link tracking completes successfully on WebKit browsers (Safari and Chrome). 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)">
```

Firefox and Internet Explorer execute the track link call and open the new page. WebKit browsers might cancel execution of the track link call when the new page opens. This often prevents track link calls from completing when using WebKit browsers.

To work around this behavior, H.25 (released July 2012) includes an overloaded track link method (`s.tl1`) that forces WebKit browsers 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 on Webkit browsers 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`).

s.tl() Parameter Reference

Variable	Description								
this	<p>The first argument should always be set either to <code>this</code> (default) or <code>true</code>. The argument refers to the object being clicked; when set to <code>"this"</code>, it refers to the <code>HREF</code> property of the link.</p> <p>If you are implementing link tracking on an object that has no <code>HREF</code> property, you should always set this argument to <code>"true"</code>.</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 <code>s.tl()</code> function is called. If you want to disable the delay, pass the keyword <code>'true'</code> as the first parameter when calling the <code>s.tl()</code> function.</p>								
linkType	<p><code>s.tl(this,linkType,linkName, variableOverrides, doneAction)</code></p> <p><code>linkType</code> 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> <tr> <th>Type</th><th>linkType value</th></tr> <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> </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 <code>null</code> if not using) This lets you change variable values for this single call. It is similar to other AppMeasurement libraries.								

Variable	Description
useForcedLinkTracking	<p>This flag is used to disable forced link tracking for WebKit browsers. Forced link tracking is enabled by default for WebKit browsers and is ignored by other browsers.</p> <p>Default Value</p> <p>true</p> <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 on WebKit browsers.</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 <code>href</code> 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 <code>href</code> 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 <a href="#"</pre>

Variable	Description
	<code>onclick="s.tl(this,'o','MyLink',null,function(){if(confirm('Proceed?'))document.location=...});return false">Click Here</code>

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 ClientCare 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="
var s=s_gi('rsid[,rsid2]'); /* see note below on the rsid */
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.

The **s_gi** function must contain the report suite ID(s) for reporting. The **tl** function accepts 3 parameters: the 'this' object (a required parameter), the link type ('e' for an exit link, 'd' for a file download, or 'o' for a generic custom link), and the link name.



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.

SiteCatalyst variables can be defined as part of custom link code.

Link Tracking Using 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:

- lnk_d = download
- lnk_e = exit

- `lnk_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 SiteCatalyst parameters (*linkTrackVars* and *linkTrackEvents*) control which SiteCatalyst 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 pass to SiteCatalyst with every file download, exit link, and custom link. The *linkTrackEvents* parameter should include each event you want to pass to SiteCatalyst with every file download, exit link, and custom link. When one of these link types occur, the current value of each variable identified is passed to SiteCatalyst.

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 SiteCatalyst variables (or events) that are defined for the current page are passed to SiteCatalyst. 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,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.

This is shown in the following example:

```
<script language=JavaScript>
function linkCode(obj) {
  var s=s_gi('rsid');
  s.linkTrackVars='None';
  s.linkTrackEvents='None';
  s.tl(obj,'d','PDF Document');
}
</script>
...
<a href="document.pdf" onClick="linkCode(this)">My Page</a>
```

The approach above does have implications for ClickMap accuracy of custom links.

With the method shown above, 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. In the example above, the file `document.pdf` is reported as a file download by automatic means. The link name `PDF Document` is reported as a file download by the custom link code. 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'); }
}
</script>
```

The `s.tl()` function determines the link type in the case of automatic link tracking for exit links and file downloads.

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.



Note: You can pass the link type and link name as additional parameters for the JavaScript function.

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.

Tracking Links with First-Party Cookies

If your JavaScript implementation uses SiteCatalyst code version H.9 or newer and relies on first-party cookies, you may need to account for this when applying tracking to the links, buttons, and other objects on your site as well.

Add `s.trackingServer` (and, where applicable, `s.trackingServerSecure`) to the list of variables in the **onClick** event handler or to your link tracking function. These variables ensure that link tracking sets or references cookies on the same data collection domain as the rest of your SiteCatalyst implementation.

For more information on first-party cookies, see the [Implementing First-Party Cookies](#) white paper.

```
onclick example:
<a href="http://www.your-site.com/some_page.php" onclick="var
s=s_gi('reportSuiteID');s.linkTrackVars='trackingServer,trackingServerSecure';s.linkTrackEvents='metrics.your-site.com';s.trackingServer='metrics.your-site.com';s.tl(this,'o','Link
Name');">Link Text</a>
function example:
<script language="javascript">
function linkTracking(obj) {
  var s=s_gi('reportSuiteID');
  s.linkTrackVars="trackingServer,trackingServerSecure";
  s.trackingServer="metrics.your-site.com";
  s.trackingServerSecure="metric.your-site.com";
  s.tl(obj,'o','Link Name');
}
</script>
<a href="http://www.your-site.com/some_page.php" onclick="linkTracking(this.href);">Link Text</a>
```

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 SiteCatalyst.

Props also let you correlate custom data with specific traffic-related events. These variables are embedded in the SiteCatalyst code on each page of your website. Through **s.prop** variables, SiteCatalyst 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: SiteCatalyst 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.

Using props vs. eVars

Props are non-persistent variables that are used to tie "traffic" data to your custom variable.

Traffic data includes visits, visitors, and page views. eVars, on the other hand, are persistent variables that are used to tie "success" events to your **custom** variable. **Custom** events include revenue, orders, leads, bookings, registrations, logins, and so forth. If you want to tie your data to traffic metrics, use a prop. If you want to tie your data to your site's custom success events, use an eVar.

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, SiteCatalyst is able to segment the users based on which pages they visited. The SiteCatalyst **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, pre-defined 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	x	x
Pathing	x	
Page View	x	x
Unique Visitors	x	x
Classifications	x	

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 any 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 SiteCatalyst 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 semi-colon. This is a legacy element for SiteCatalyst. 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. In order to have the maximum flexibility in your reporting, it is best to leave this blank and use **Classifications** in SiteCatalyst 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.



Note: A single image request cannot contain a numeric or currency event higher than 2,147,483,648 or a counter event higher than 2,000,000,000.

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 the SiteCatalyst Admin Console.

Email Campaign Tracking

Companies can use SiteCatalyst to determine the success of an email campaign.

Companies often develop costly email campaigns to market their products. Companies send bulk emails to customers with the expectation that some customers will access the site and a conversion will result. SiteCatalyst can report 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.

Metric	Description
Purchases	Displays the number of purchases resulting from the email.
Segmentation by Subcategory	Displays comparison data between different types of users.

Modifications to the HTML email body and the JavaScript library are required to capture the key metrics shown above.

Implementing Email Campaign Tracking

There are several steps to follow in order to successfully display email campaign analysis data in SiteCatalyst.

The steps are listed and described as follows:

1. Create unique tracking codes.

Though each tracking code must be unique, it does not need to be more than a number. The list below shows examples of unique tracking codes.

- Email Opens: 112233A
- Promo Link 1: 112233B
- Promo Link 2: 112233C

2. Add query string parameters to HTML email links.

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 may 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 should be added to the link above:

```
<a href="http://www.mycompany.com/index.asp?sc_cid=112233B&sc_v1=SUBCATEGORY">Visit our
  home page</a>
```

3. Update the JavaScript library.

Altering SiteCatalyst JavaScript code in the JavaScript file, `s_code.js`, allows SiteCatalyst to 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.

1. 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 that 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('sc_cid')
}
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* and `sc_v1` is copied into *eVar1*.



Note: Only the first call to `getQueryParam` is required to capture click-throughs. Contact your Adobe **Implementation engineer** to implement this function and to ensure that your version of the JavaScript file contains the `getQueryParam` plug-in.

2. 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:

- The query string parameters "`sc_cid`" and "`sc_v1`" must be visible in the URL on the final landing page, otherwise no click-through conversion are recorded.
 - Examples of query string parameters are "`sc_cid`" and "`sc_v1`." Any query string parameter can be used and captured by the `getQueryParam` plugin. 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` and `eVar1`.
4. Use SAINT to classify campaign tracking codes (optional).

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. The following steps outline the process required to set up an email campaign in SiteCatalyst. For more information, see the [SiteCatalyst SAINT Classifications Guide](#).

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 ClientCare about turning on pathing for a **Custom Traffic** 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 will display as shown below:

```
s.prop1="Home Page : Spring Promo Email"
```

When pathing is enabled for the **custom traffic** variable, you can use **Path** reports (**Next Page Flow**, **Fallout**, etc.) to see visitor navigation from the landing page.

Tracking External Email

SiteCatalyst can report campaign analysis data in several additional key metrics with configurations made by Adobe ClientCare.

Campaign Type	Description
Emails Sent	Displays the total number of emails sent by the company. This metric is provided by the email vendor.
Emails Delivered	Displays the total number of emails delivered to email servers. This metric is provided by the email vendor.
Emails Opened	Displays the number of times the visitors opened the email.

For more information on tracking email campaign analysis data, contact Adobe ClientCare. Tracking email campaign analysis data is considered on a case-by-case basis.

Products

Products refer to one of many values contained in the products variable.

The **Products** value is used for tracking multiple products and product categories. It is also used for purchase quantity, purchase price, and event serialization and merchandising.

When Should the Product Variable be Set?

Set the *products* variable on pages when you need to track information about a product or multiple products.

These types of pages generally show product information to the user. Some specific examples might include (but are not limited to) product detail pages, shopping carts, promotion pages, and landing pages. The most common use of *products* variable is in the stages of the purchase process.

Setting a Product with an Event

Adobe best practice is to set the *products* variable on the same page as an event.

This ties the product(s) on the page to the event for reporting in SiteCatalyst. While any event can be used to tie data to products, the most common events are those associated with the shopping cart and purchase. These events include:

- Product views (**prodView** event)
- Cart additions (**scAdd** event)
- Checkouts (**scCheckout** event)
- Purchases (**purchase** event)

An example of each is included below:

Product Views

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

```
s.products=" ;PRODUCT_NAME "
```

Cart Additions

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

```
s.products=" ;PRODUCT_NAME "
```

Checkout Page

```
s.events="scCheckout "
```

```
s.products=" ;PRODUCT_NAME "
```

Purchase Page



Note: Multiple events and eVars are delimited with a pipe (|). In addition, multiple product strings are delimited with a comma (as shown in the Purchase Page example above).

Setting a Product without an Event

Adobe recommends setting an event every time the *products* variable is used.

If omitted, the system tracks a **product view** (prodView) event by default.

Field Definitions of the Product String

List of field definitions contained in the product string and their definitions.

Field	Definition
product_name	The identifier used to track a product. This identifier is used to populate the Products report in SiteCatalyst. Be sure to use the same identifier through the checkout process.
units	The number of units purchased. This field must be set with a purchase event to be recorded.
total_cost	Refers to the combined cost of the total units purchased, not to the unit cost. This field must be set with a purchase event to be recorded.
events	Custom events associated to a particular product; for example, discounts.
eVars	Values associated with a specific product, such as merchandising category.

Sample Product Strings

Each of the following examples shows the product string with a different configuration.

- **s.products=";ABC123"**

- **s.products=";ABC123;;ABC456"**

- **s.products=";ABC123;1;10"**

- **s.products=";ABC123;1;10;;ABC456;2;19.98"**

- **s.events="event1"**

s.products=";ABC123;;;event1=1.99"

- **s.events="event1"**

s.products=";ABC123;1;10;event1=1.99"

- **s.events="event1"**

s.products=";ABC123;1;10;event1=1.99;;ABC123;2;19.98;event1=1.99"

- **s.events="event1,event2"**

s.products=";ABC123;1;10;event1=1.99|event2=25"

- **s.events="event1,event2"**

s.products=";ABC123;1;10;event1=1.99|event2=25;evar1=Super Fast Shipping"

- **s.events="event1,event2"**

s.products=";ABC123;1;10;event1=1.99|event2=25;evar1=Super Fast Shipping|evar2=3 Stars"

- **s.events="event1,event2"**

s.products=";ABC123;1;10;event1=1.99|event2=25;evar1=Super Fast Shipping,
;ABC456;2;19.98;event1=1.99|event2=100;evar1=really slow shipping"

- **s.events="event1,event2,event3"**

s.products=";ABC123;1;10;event1=1.99|event2=25;evar1=Super Fast
Shipping;ABC456;2;19.98;event1=1.99|event2=100;evar1=really slow shipping;;;event3=2.9;evar3=20% off"

- **s.events="event1,event2,event3"**

s.products=";ABC123;;;ABC456;2;19.98;event1=1.99|event2=100;evar1=really slow shipping;;;event3=2.9;evar3=20% off"

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 SiteCatalyst. The tremendous insight it brings is critical to businesses looking to understand visitor traffic patterns.

Out-of-the-box SiteCatalyst 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.

For a detailed understanding of these reports, a training video on "Path Analysis" in the Help section of SiteCatalyst is available. Other knowledge base articles on the subject can be found there as well. This section focuses on the implementation issues of pathing and extending pathing functionality to be able to segment your path reports.

Enabling Pathing on a prop

Though SiteCatalyst pathing reports are available out-of-the-box for pages, pathing can also be enabled for prop variables which are **custom traffic** variables.

This setting is not enabled for any props by default because it is only appropriate in certain cases. Enabling pathing on a prop can only be done by Adobe Support (ClientCare, Account Managers, and Consultants).

Depending on your contract, there may be an additional fee to enable pathing per prop. Your Account Manager or ClientCare representative can let you know of any applicable additional fees.

You first need to clearly understand the business questions that are being asked. Enabling pathing for a custom prop may be necessary to answer that question.

Below are a few examples of a business question that can be solved by turning on 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?

You are trying to segment your pathing reports by various dimensions (campaigns, user type, and so forth). In the following sections, we provide examples of how to use pathing on props to answer these business questions.

It is important to note that though this data can be achieved by enabling pathing on props and structuring the data in a specific way as outlined below, that the optimal way to segment pathing reports is using Adobe Discover. Discover has been specifically designed to provide 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 [Adobe Discover documents](#).

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.

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 SiteCatalyst.

- 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. SiteCatalyst 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 ClientCare 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 ClientCare 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."

Using Implementation Plug-ins

SiteCatalyst JavaScript plug-ins are programs or functions that perform several advanced functions.

These functions are listed below:

- Retrieve query string parameter values
- Detect Flash
- Write cookies to store session data
- Meet other business requirements

These plug-ins extend the capabilities of your JavaScript file to give you more functionality that is not available with a basic implementation. Even though Adobe offers several plug-ins, only a few of them are implementation-specific.

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 SiteCatalyst JavaScript file (*s_code.js*).

```
/* Plugin Config */
s.usePlugins=true
function s_doPlugins(s) {
  /* Add calls to plugins here */
}
s.doPlugins=s_doPlugins
```

Renaming the *doPlugins* Function

The *doPlugins* function is typically called *s_doPlugins*. In certain circumstances, (usually when more than one version of SiteCatalyst 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
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 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 ...
//
```

Implementation Plug-ins

Descriptions of the available plug-ins that you can use in a SiteCatalyst implementation. These plug-ins can be downloaded from SiteCatalyst at **Help > Help Home**.

Complete documentation for each of these plug-ins is available there. 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.

Plug-in	Description
apl Plug-in Utility	Appends a value to any delimited lists.
getQueryParam	Returns the value of the query string parameter found in the current URL. If no query string parameter is found with that value, an empty string is returned.
getValOnce	Forces a variable to be populated only once within a single session or time period. The most common reason for doing this is to keep campaign click-throughs from being inflated.
getPercentPageViewed	Records the portion of a page (0-100%) that the user has viewed and passes it into a variable on the next page view.
Time Parting	Captures time of day, day of week, and weekday/weekend status in SiteCatalyst reports for segmentation by time.
detectRIA	Detects the versions of Adobe Flash and Microsoft Silverlight installed in the users' browsers, so that SiteCatalyst variables capture this information.

Plug-in	Description
daysSinceLastVisit	Determines the number of days since the user last visited your site.
getNewRepeat	Captures new/repeat visitor status of users on your site.
getAndPersistValue	Captures the value of a variable and passes it into another variable on every page view for a designated period of time.
getPreviousValue	Places the value of a selected variable into a cookie and then captures this value into another variable on the next page view.
getVisitNum	Uses a cookie to determine the total number of visits that the user has made to your site.

Implementation Steps

Obtain the JavaScript plug-in through Adobe ClientCare or your Adobe Consultant. Include the plug-in function in the **Plug-ins** section of the Adobe JavaScript file. See example below:

```
s.usePlugins=true
function s_doPlugins(s) {
  /* Add calls to plugins here */
}
s.doPlugins=s_doPlugins
/***** PLUGINS SECTION *****/
/* You may insert any plugins you wish to use here. */

[Insert Plug-in Code Here]

...Rest of the JS File...
```

Call the plug-in from within the *doPlugins* section of the JS file as shown below:

```
s.usePlugins=true
function s_doPlugins(s) {
  /* Add calls to plugins here */
  [Insert Calls to Plug-ins Here]
}
s.doPlugins=s_doPlugins
/***** PLUGINS SECTION *****/
/* You may insert any plugins you wish to use here. */

[Insert Plug-in Code Here]

...Rest of the JS File...
```


DigitalPulse Debugger

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

When executed in your browser, it shows the image requests that transmitted data from that page into SiteCatalyst, Test&Target, Recommendations, and/or Survey, 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.

Automatic Setup

Configure DigitalPulse Debugger automatically by dragging a button from this section to your **Bookmarks** menu.

Drag the following button to your bookmarks toolbar or to the desired location in your **Bookmarks** menu.

DigitalPulse Debugger

If you are using Internet Explorer, a notification displays prompting you to add this bookmarklet. Click **Yes**.



Note: Clicking the **DigitalPulse Debugger** button launches the DigitalPulse debugger for this specific page. In order to use it successfully on other sites, you must create a bookmark.

Manual Setup

Copy JavaScript code to your clipboard and then configure DigitalPulse Debugger in the browser of your choice.

1. Copy the following text to your clipboard (triple-clicking might help in highlighting all the text):

```
javascript:(function(){var s=document.getElementsByTagName('script').length-1;var d=document.createElement('script');d.src='http://www.digitalspulse.com/dpdebug.js';document.getElementsByTagName('script')[s].parentNode.appendChild(d)});
```

2. Complete the steps appropriate for the desired browser:

Mozilla Firefox

Instructions to create a bookmark for the DigitalPulse Debugger from within Mozilla Firefox.

1. Right-click the bookmarks sidebar, then click **New Bookmark**.
2. In the **Name** field, specify "DigitalPulse 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 [Manual Setup](#).
4. Select **Load the Bookmark in the Sidebar**, if desired.
5. Click **Add**.


Google Chrome

Instructions to create a bookmark for the DigitalPulse Debugger from within Google Chrome.

1. Right-click the **Bookmarks** bar (depending on your Chrome settings the **Bookmarks** bar might be on the **New Tab** page), then click **Add Page**.
2. In the **Name** field, specify "DigitalPulse Debugger" as the name for the new bookmark.
3. In the **URL** field, paste the code that you copied to your clipboard as explained in [Manual Setup](#).
4. Browse to and specify the location you want to save the bookmark.
5. Click **OK**.

Microsoft Internet Explorer

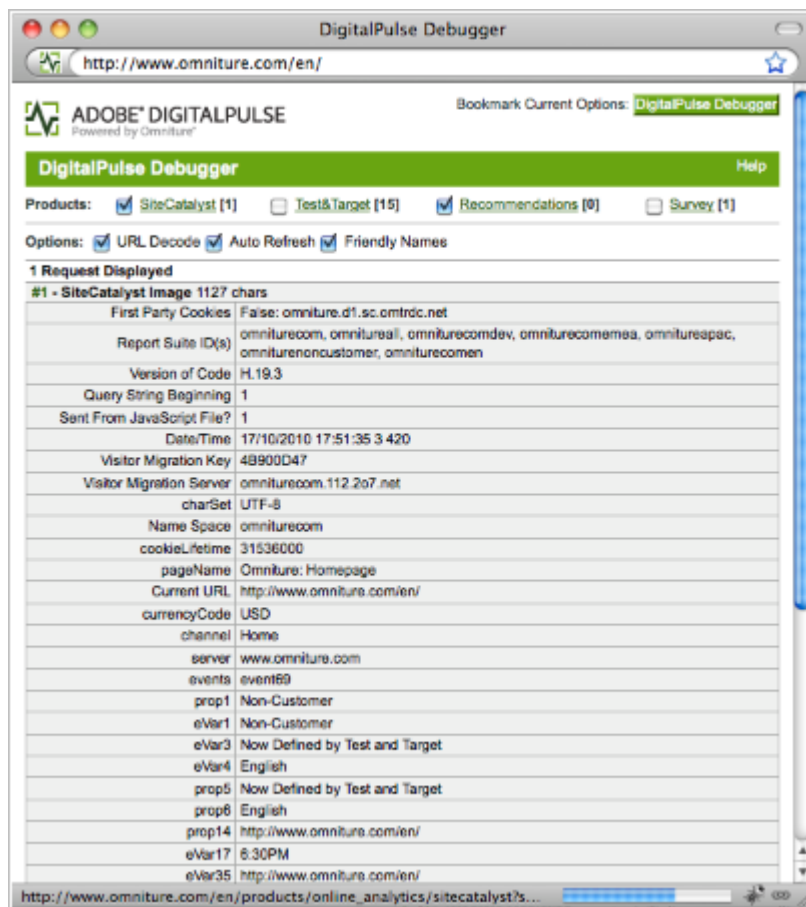
Instructions to create a bookmark for the DigitalPulse Debugger from within Microsoft Internet Explorer.

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 labeled "Omniture Help."
2. Right-click the "Omniture Help" bookmark, then click **Rename**.
3. In the **New Name** field, specify "DigitalPulse 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 [Manual Setup](#).
6. Click **OK**.

Launching DigitalPulse Debugger

After installing the DigitalPulse Debugger, you are ready to use it to view the data being collected from your site.

1. From your browser, click the bookmark or bookmarklet to launch the DigitalPulse Debugger in a new window.



2. View the data.

Dynamic Variables and Query String Parameters

When using packet monitors or older versions of the debugger, you will find query string parameters instead of variable names.

The following table is a comprehensive list showing all query string parameters with their associated variable and explanation in alphabetical order:

Parameter	SiteCatalyst 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)

Parameter	SiteCatalyst Variable	Report Populated	Description
c1-c75	s.prop1-s.prop75	All Custom Traffic reports	Traffic variables used in custom traffic reporting
cc	s.currencyCode	None	The type of currency used on the site
ce	s.charSet	None	The character encoding of the image request
cdp	s.cookieDomainPeriods	None	Indicates the number of periods in a domain for cookie tracking; manually set.
ch	s.channel	Site Content Site Sections	The Site Sections variable used in traffic reporting
ct	None	Visitor Profile Technology Connection Types	Connection Type (Modem, LAN, etc; can only populate in IE browsers)
events	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
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)
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
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	s.pageName	Site Content Pages	The page's designated name in reporting

Parameter	SiteCatalyst Variable	Report Populated	Description
pageType	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
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	s.products	Products Products	Products variable used in conversion reporting
purchaseID	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 × height)
server	s.server	Site Content Servers	The page's server; used in traffic reporting
state	s.state	Visitor Profile Visitor State	Specifies the state as defined by the variable; deprecated with v15
t	None	None	Browser time information- "[d/m/yyyy] [hh:mm:ss] [weekday] [time zone offset]"
v	None	Visitor Profile Technology JavaScript	JavaScript 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

Parameter	SiteCatalyst Variable	Report Populated	Description
vid	s.visitorID	None	The visitor's unique ID
vmk	s.vmk	None	Visitor migration key; used to migrate from third-party to first-party cookies
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

Packet Analyzers

Packet analyzers let you view the data sent by your implementation to Adobe Data Collection Servers.

Similiar 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 SiteCatalyst requests, filter outgoing requests using "b/ss".

In very rare cases, the debugger will report an image request although no request makes it to our SiteCatalyst 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
HttpFox	HttpWatch	Chrome Developer Tools	Charles
Tamper Data		Firebug Lite	Fiddler
HttpWatch			Wireshark
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.

Testing and Validation

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]
```

Deployment Validation

The SiteCatalyst code can be placed anywhere inside BODY tags (`<BODY></BODY>`) of a well-formed HTML page.

Browsers do not interpret image requests placed inside HEAD tags (`<HEAD></HEAD>`) correctly. This can cause the code to function incorrectly. We recommend 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.

- Do not locate the code within graphical elements or tables, and ensure that the code does not move any elements (images, tables, and so forth) unintentionally.
- Place the HTML and reference to the .JS file as high on the page as possible in order to measure more page views before a visitor clicks to another page.
- Place the code only within the `<td></td>` tags, if placed within a table.
- The code that sets the variables must occur prior to the reference to the .JS file.
- Make certain that the report suite IDs (shown below in the sample as "sampleco") are set correctly. The code should be supplied by the Adobe Consultant with a report suite ID in place. You may need to change this report suite ID at some point. Always ensure that the ID is correct.

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 SiteCatalyst .JS 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 SiteCatalyst code into a default HTML page to determine if the error is occurring because of other page elements/objects.

```
<html><head></head><body>
...paste SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst.



Note: Multiple page records cannot be combined within SiteCatalyst 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 [Link Tracking](#).

Custom Variables

List of **custom** variables used in SiteCatalyst.

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?

Variable	Description
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 SiteCatalyst 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 SiteCatalyst and verifying all variables (*page name, channel, server, events, campaign*, econversion variables, custom traffic variables, *products*, and all other SiteCatalyst 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 uses Adobe DigitalPulse or other auditing solution to ensure that all pages contain the SiteCatalyst code.

DigitalPulse is a tool that allows the client or consultant to crawl the site that has been implemented to validate that SiteCatalyst beacons are firing on each page. The Adobe Consultant can assist in suggesting the most appropriate methods of crawling the site within DigitalPulse or another implementation auditing tool.

7. The client faxes (or signs online) an Implementation Acceptance and Agreement for the appropriate site(s).
8. After the acceptance has been received, the Adobe Consultant enables the Adobe Best Practices - Implementation Verification certification within the interface.
9. 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 SiteCatalyst 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 SiteCatalyst 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 by SiteCatalyst.
- 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 SiteCatalyst 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 (SiteCatalyst de-dupes the orders based on *purchaseID*).

Common Syntax Mistakes in SiteCatalyst

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 SiteCatalyst code are listed in the following sections.

Putting SiteCatalyst Code in the Head Tag

SiteCatalyst code creates an image object, a non-visible image that does not show up on your page.

Previously, a common implementation practice was to place the SiteCatalyst JavaScript code between the `<head>` and `</head>` tags. By placing the code between these tags it prevented the 1×1 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 SiteCatalyst 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 SiteCatalyst, and users may wonder why certain reports and tools, including **ClickMap**, aren't recording data.

The SiteCatalyst 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 SiteCatalyst with Test&Target, the `s_code` include file must be placed at the bottom of the page. The following example shows correct placement of the SiteCatalyst code:

```
<html>
<head></head>
<body>
<!-- SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst, 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 that SiteCatalyst offers.

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.

This variable is explained in detail in the [SiteCatalyst Implementation Manual](#) and the online Knowledge Base. Here are a few extremely easy-to-make mistakes that can wreak havoc 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, SiteCatalyst 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 SiteCatalyst to read this as shown below:

Category 1: widgets

Product 1: large widget, 40x40

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. SiteCatalyst 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 [™], ©, 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. SiteCatalyst treats these as distinct values for the **s.pageName** variable. The SiteCatalyst 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 SiteCatalyst Code

Adobe offers some best practices for replacing your SiteCatalyst 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
<code>prop1</code>	<code>s.prop1</code> (uses "s.")
<code>s.evar1</code>	<code>s.eVar1</code> (uses correct capitalization)
<code>s.pagetype='errorpage';</code>	<code>s.pageType='errorPage';</code> (uses correct capitalization)
<code>s.tl(this,o,pageA)</code>	<code>s.tl(this,'o','pageA');</code> (correct usage of single quotes)
<code>s.events='event1'; s.events='event2';</code>	<code>s.events='event1,event2';</code> (correct format)
<code>s.pageName="John"</code> (smart quotes on)	<code>s.pageName="John"</code> (smart quotes off)
<code>s.products="shoes,UX4879,1,19.99"</code>	<code>s.products="shoes;UX4879;1;19.99"</code> (uses semicolons, not commas)
<code>s.products=";MacBook Air;1;\$1999.99"</code>	<code>s.products=";MacBook Air;1;1999.99"</code> (does not use dollar signs)
<code>s.products=";Nikon SB-600 Speedlight Flash for Nikon Digital SLR Cameras;1;229.9489183"</code>	<code>s.products=";Nikon SB-600 Speedlight Flash for Nikon Digital SLR Cameras;1;229.95"</code> (round or truncate long prices)
<code>var s_account="rsid1, rsid2"</code>	<code>var s_account="rsid1,rsid2"</code> (no space between report suite IDs when doing multi-suite tagging)
<code>s.events="event1, event2"</code>	<code>s.events="event1,event2"</code> (no space between event IDs when doing multi-event tagging)
<code>s.products="product name"</code>	<code>s.products=";product name"</code> (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).

Optimizing your SiteCatalyst Implementation

SiteCatalyst 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 SiteCatalyst used during metrics collection. If the SiteCatalyst 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 SiteCatalyst overhead.

Variable Length

The length of SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst customers have SiteCatalyst 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 SiteCatalyst 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. Customers should ensure that https is used only where necessary.

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.

The SiteCatalyst 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.

JavaScript File Location and Concurrency

Placing the call to the JavaScript library file at the top of the page ensures that the SiteCatalyst 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 SiteCatalyst 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.

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), SiteCatalyst uses the URL of the page. Because the page name is central to your SiteCatalyst reports, make sure that all parties in your organization agree on a strategy before you implement SiteCatalyst.

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 SiteCatalyst variables. Many companies find that most SiteCatalyst 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
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 SiteCatalyst users in your organization. Examples of page names include Home Page, Electronics Department Home, and Sports : Local Sports : High School.

Most SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst 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 SiteCatalyst page names are not affected. If a page name changes in SiteCatalyst, it becomes a new page and is not connected with the old page name, regardless of the associated URL.

Optimum Path Engine

The **Pathing** reports in SiteCatalyst use the value of the *pageName* variable (or the URL if *pageName* is blank) to show the paths people take through your site.

While **path** reports are shown for page names by default, you can enable **Path** reports for other variables in SiteCatalyst. You can see **path** reports by site section, sub-section, or detailed content section.

Changing Page Names in SiteCatalyst

The **Page Naming Tool** can be used to change page names as they appear in SiteCatalyst to give pages friendlier names.

This tool lets you change the displayed page name rather than the value of the *pageName* variable in order to improve the readability of your reports. Because most SiteCatalyst users expect the *pageName* variable to match the value shown in SiteCatalyst reports, using this tool may cause confusion within your organization. This tool does not combine pages, even if they are given identical names.

If you want to modify the page name, Adobe strongly recommends that you edit the existing page name implementation rather than using this tool. However, if you want to use the **PageName** tool, do so with caution, and only after you fully understand the effects of doing so. Create a log of changes to the page name when using the **Page Naming Tool**.

The **Page Naming Tool** is most often used from the **Pages** report. Click a line item in the report, and a drop-down list displays. By default the **Page Naming Tool** displays (**Rename Page**). Just retype the name you have selected for the page.

Using the Name Pages Tool

The **Name Pages Tool** is used to add friendly names to the pages of your website so that when the page name is displayed in a report, you can view the data and the corresponding Web page name in an easy-to-read format. To access the **Name Pages Tool** click **Admin > Name Pages**.

The order pages appear in the **Name Pages Tool** is the same order they appear in the **Pages** report for the current month.

The format of the **Name Pages Tool** can be misleading because it directs the user to believe that each page in the system has a unique URL with one associated page name, which is not the case. In fact, you may have multiple URLs associated with a single page name.

Only administrators or groups that have specifically been given access to the **Page Naming Tool** can use it to rename pages. It is important that only a few people have access to the **Page Naming Tool** in order to avoid problems with page renaming. If a page is renamed, you may not recognize it in the **Pathing** or **Pages** reports.

SiteCatalyst does not record when a user changes a page name, but does record when a user views the **Name Pages Tool**. To view the **Page Renaming Log**:

1. Click the **Permissions** tab.
2. Access the **Security and Access Log**.
3. Under **Event Type**, choose **Tool Viewed**.
4. Under **Event**, choose **Name Pages Viewed**.

Storing Page Names in SiteCatalyst

Every **Page View** recorded in SiteCatalyst came from a page with a specific page name. This is either the URL of the page or the value of the *pageName* variable. Adobe recommends that you use the *pageName* variable to name pages because it offers the most flexibility in defining a page.

When SiteCatalyst receives a page view, one of the first steps taken is to update the database table containing the page names. The following example shows the organization of the database table:

Key (Hash)	Page Name	URL
cfe6e34	Home Page	http://mysite.com
d2ed34	http://mysite.com/sitemap.asp	http://mysite.com/sitemap.asp
9a2168	Partial Site Map	http://mysite.com/sitemap.asp?id=35

The Key column in the preceding table is created from the original page name as it enters SiteCatalyst. The Page Name column contains the current page name value, which may be altered via the **Page Naming Tool**.

Additional Notes

- The only way to undo the page renaming is to revert to the original page name sent to SiteCatalyst.
- Keep a log of changed pages in the Notes section of the **Pages** report.
- Put a reference in the Notes section of where the log should be kept.

Unique Value Limits

Adobe recommends values be kept within a certain monthly threshold.

SiteCatalyst can easily track millions of products and transactions that happen on the site every day. Adobe recommends the number of unique *pageName* values be kept within a monthly threshold of 250,000 unique page names. If the threshold is maintained in the reports, the interface typically responds more quickly.

Dynamic Variables

Dynamic variables are shortened identifiers that correspond to Adobe SiteCatalyst variables. Dynamic variables let you copy values from one variable to another without typing the full values multiple times in the SiteCatalyst image requests on your site.



Note: The following information is written primarily for developers implementing SiteCatalyst. If you have questions about implementing dynamic variables, your organization's supported users should work with the appropriate development resources in your organization. You can also contact Adobe ClientCare.

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 SiteCatalyst 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.

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

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 [Dynamic Variable Abbreviations](#).

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.

The following information applies to dynamic variable use in SiteCatalyst:

- Dynamic variables work with all versions of SiteCatalyst code.
- 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 SiteCatalyst 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. For more information on variable limits, see the [SiteCatalyst Implementation Guide](#).

Dynamic Variable Examples

Examples of dynamic variables you can use in SiteCatalyst.

```
s.eVar7="D=pageName" // captures the pageName value in
eVar7

s.prop15='D=v50+":'+c6' // concatenates eVar50:prop6 into
prop15

s.prop5=s.eVar5="D=g" // passes the page URL into both prop5
and eVar5

s.eVar20="D=v0" // captures the campaign in eVar20
```

Note that the D=[variable] value should be in quotes. The SiteCatalyst code treats this as a string. The string will be URL encoded when passed into SiteCatalyst (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.

Other Uses for Dynamic Variables

A primary use for dynamic variables is the capture of HTTP headers, including the values of cookies passed along with them.

For example:

```
s.prop1="D=User-Agent" // captures the user-agent string in
prop1

s.prop25="D=s_vi" // captures the s_vi (visitor ID) cookie value in prop25
```

Using dynamic variables in this manner does not cause any parsing of header or cookie values. Whatever is present in these fields is what you get in your reporting. Any other header can also be captured using the header name (for example, **Referer**, **Accept-Language**, and so forth).



Note: Capturing the visitor ID may have limited usefulness in many cases. If you receive more than 500,000 monthly unique visitors, you will exceed the maximum number of unique values allowed for whatever variable you choose to receive this data.

Being able to capture cookie values can be helpful in CNAME (first-party cookie) SiteCatalyst implementations where you have your own site cookies to capture for reporting. For example, if you capture a campaign ID and store it in a cookie named mktg_camp using JavaScript, you can then read that cookie and set the value of the cookie into eVar1 by putting the following on each page:

```
s.eVar1="D=mktg_camp"
```

This causes the campaign ID, found in the mktg_camp cookie to be passed into SiteCatalyst on each page.

Dynamic Variable Abbreviations

Dynamic Variables with their abbreviations.

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. The dynamic variable usage must match the parameter in the

given request. For example, 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.

Abbreviation	Variable
pageName (or gn)	s.pageName
server (or sv)	s.server
pageType (or gt)	s.pageType
ch	s.channel
c1 - c75	s.prop1 - s.prop75
h1 - h5	s.hier1 - s.hier5
v0	s.campaign
state	s.state
zip	s.zip
events (or ev)	s.events
products (or pl)	s.products
purchaseID (or pi)	s.purchaseID
v1 - v75	s.eVar1 - s.eVar75
ce	s.charSet
cc	s.currencyCode
cdp	s.cookieDomainPeriods
cl	s.cookieLifetime
r	s.referrer (referring URL)
g	s.pageURL (current URL)
vid	s.visitorID
xact	s.transactionID

The following variables are typically set directly by the SiteCatalyst JavaScript file. These variables do not have a corresponding, customizable JavaScript variable. However, they can still be captured using dynamic variables.

Abbreviation	Variable
s	Screen resolution (width x height)
c	Screen color depth (in bits)
j	JavaScript version (e.g., 1.3)
v	Java enabled (Y or N)
bw	Browser Width (in pixels)
bh	Browser Height (in pixels)
t	Web browser time info (DD/MM/YYYY hh:mm:ss weekday timezone)
k	Cookie support (Y or N)

Abbreviation	Variable
ct	Connection Type ("modem" or "lan")
hp	Current page is browser home page (Y or N)
p	Comma-separated list of Netscape plug-in names

Identifying Unique Visitors

Adobe uses a cookie to track unique browsers/devices.

When a user visits your site, a persistent cookie, called `s_vi`, is set in the visitor's browser. This cookie is set on the specified data collection domain. This domain ends with `2o7.net` if your site uses third-party cookies, or is your domain with a special sub-domain (`metrics.yoursite.com`) if your site uses first-party cookies.

Fallback Visitor Identification Method

JavaScript H.25.3, released January 2013, contains 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 first-party cookie is created 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, especially for sites using third-party cookies. If the `s_vi` and the `s_fid` cookies cannot be set, visitors are identified using a combination of IP address and User Agent.

SiteCatalyst 15 visits total includes all visitors that are identified by the `s_vi` cookie or by using a fallback method.

SiteCatalyst 14 visits total includes only visitors that are identified using the `s_vi` cookie. Visitors identified by the fallback method are not included in the visits total.

Wireless Device Identification

Because wireless devices do not normally accept cookies, a more reliable method needs to be used to uniquely identify wireless devices other than cookies. Adobe has identified a number of HTTP headers that uniquely identify a small number of wireless devices. Those headers often include the device phone number (or a hashed version), or other identifiers.

Under the Adobe current mobile tracking method, the percentage of uniquely identifiable devices is approximately 10% on non-carrier specific sites. These devices have one or more of the headers that uniquely identify the device, and all Adobe data centers automatically use those headers in lieu of a **Visitor ID**. There is no special effort during implementation that is required to take advantage of these headers for device and visitor identification.

In addition, beyond **visit** and **visitor**, the accuracy of other reports/metrics are heavily affected by the inability to uniquely identify a mobile device. Specifically, pathing reports and any commerce reports depending on persistent variable behavior (for example, **eVars**, **campaigns**) wouldn't be populated

Adobe assumes a device is a mobile device if the `/5/` appears in the path. If a mobile page contains a beacon URL with `/1/`, Adobe ignores all headers and tries to use a cookie to identify the visitor.

If the special headers that Adobe has identified are not available for a device, the fallback method is used to identify Visitors: the IP address and User Agent string. Because many IP addresses and devices can be mapped to the same Wireless Gateway, it is common that multiple devices appear to be the same device. Adobe does not include this traffic in the pathing reports, or in calculating the **Visits** metric.

Unique Visitors reports provide the ability to distinguish between visitors that accept a **visitor ID** cookie and those that don't. Mobile visitors with the appropriate **visitor ID** header are treated as if they accept cookies, while those without headers are treated as if they block cookies in those reports.

The following list of headers is used to identify wireless devices. The algorithm for processing the headers is to extract the HTTP header key (the name of the header, such as, "X-Up-Calling-Line-ID"), trim out all non alpha (A-Z and a-z) characters, convert the header key to lowercase, and then compare the end of the key to the ones in the following table to find a match. 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").

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

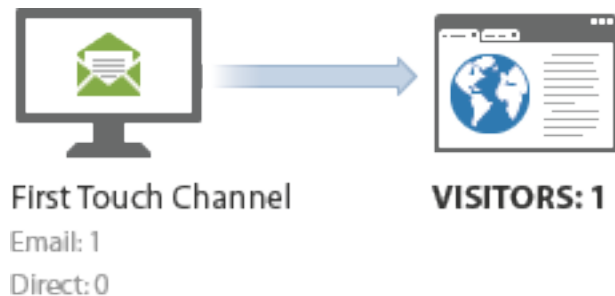
Cross-Device Visitor Identification

Cross-device visitor identification is a SiteCatalyst feature that helps you connect visitors across multiple devices.

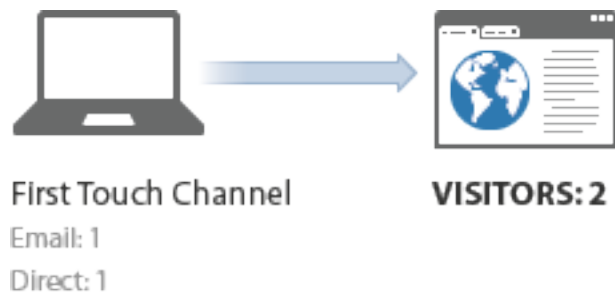
This feature works by associating the visitor profiles across each device where a user is uniquely identified. After association, the visitor profile records are merged and visits from each device are considered to be from the same user.

Multiple Devices

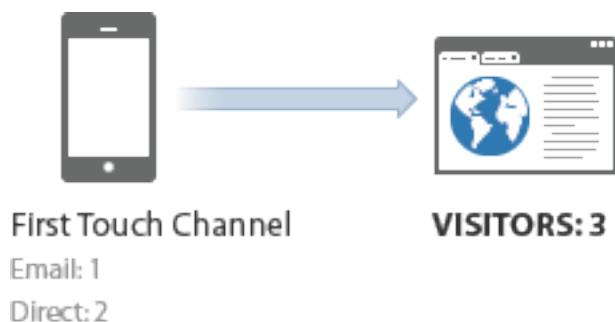
With the increasing number of devices used by the same visitor, it can be difficult to accurately gauge unique visitors and correctly attribute first-touch marketing channels. For example, a potential customer clicks a link in a marketing email and views your site for the first time:



This customer is considered a unique visitor and the first-touch email channel is incremented. The next day, this same customer visits your site from a laptop at his office:



Since the device is new, the customer is considered a unique visitor and the first-touch direct channel is incremented. A few days later, this same customer visits from a mobile device:



The customer is again treated as a unique visitor.

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.

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 customer clicks a link in a marketing email and visits your	1	-	1	1	8

Server Call	Action	Visitor ID Cookie	Visitor ID Variable	Effective Visitor ID	Visit Page Number	Visit Number
	site from home computer. This customer has visited your site 7 other times in the past.					
2-8	Visits 7 additional pages on your site.	1	-	1	2-8	2-8
9	Authenticates.	1	CID1	CID1	1	8
10	Opens site from laptop at office.	2	-	2	1	1
11	Authenticates.	2	CID1	CID1	1	9
12	Views an additional page.	2	CID1	CID1	2	9

Visitors

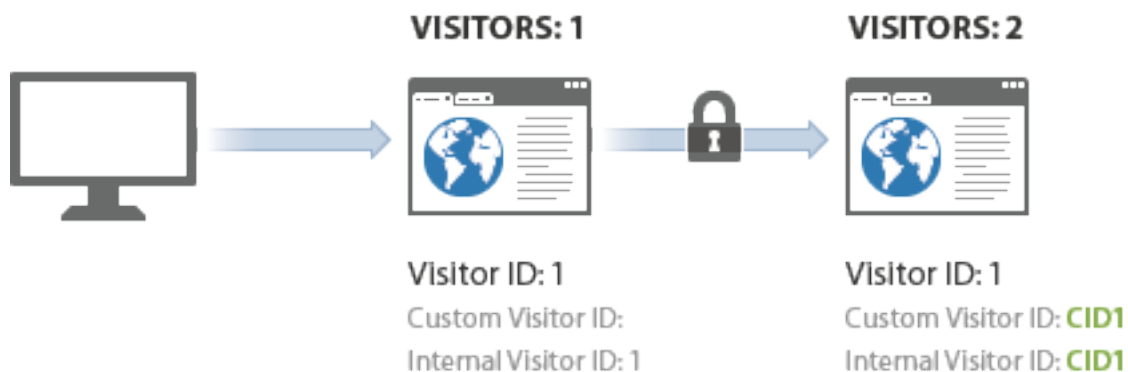
SiteCatalyst 15, DataWarehouse, and **Discover** count each unique effective **visitor ID** as a unique visitor.

If you look at the [previous table](#), this occurred 3 times: at hits 1, 9, and 10. SiteCatalyst 14 does not count an additional visitor for server call 11 if it is on the same day as server call 9. 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.

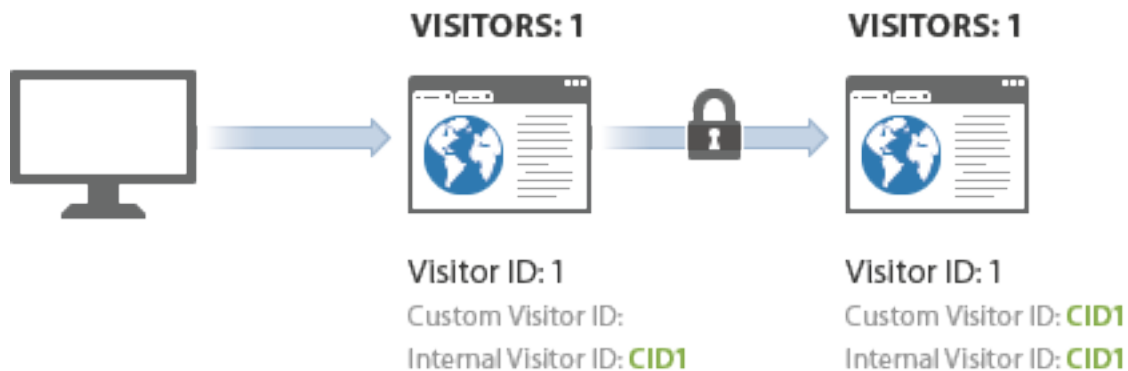
This can increase the number of unique visitors you see when cross-device visitor identification is enabled. The visitor might be counted twice on the same visit: once for the initial visit and again after the user is authenticated.

When a new visitor views your site, the **visitor ID** cookie is populated and stored in the **s_vi** cookie. On the data collection server, a new visitor profile is created for this **visitor ID** cookie, and the effective **visitor ID** on the profile is set to match the **visitor ID** 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

SiteCatalyst 14 and 15, DataWarehouse, and **Discover** count 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, 10, and 11. Similar to visitors, this value returns to normal after the initial association because the **Visit Page Number** is not reset back to 1 due to a change in the effective **visitor ID**.

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 11 and 12. Even though server call 10 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 data, it is based on the IP address in each hit. This could result in multiple geo values if the IP address changes for the same visit.

Persisted Dimensions (eVars, Marketing Channels, Campaign)

When visitor profiles are merged after being associated with the same **visitor ID** variable, attribution is not changed in the historical data set.

This often results in additional attribution to the direct channel because your site is accessed directly before the customer is authenticated.

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.

SiteCatalyst and Redirects

SiteCatalyst 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 SiteCatalyst 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, SiteCatalyst 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. SiteCatalyst 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.

2. 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/`.
3. 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 SiteCatalyst data collection is lost because the browser clears the referring URL. Thus, the original search information used in the SiteCatalyst reports (for example, **Referring Domains**, **Search Engines**, **Search Keywords**) is lost.

Implementing Redirects for SiteCatalyst discusses how to leverage SiteCatalyst 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 for SiteCatalyst

In order to capture SiteCatalyst data from redirects, four minor alterations need to be made to the code that creates the redirect and the SiteCatalyst .JS 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

Adobe ClientCare can provide you with a JavaScript file that contains code that will override the default functionality SiteCatalyst uses to capture the referrer from the HTTP header.

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.

```
H Code
<!-- SiteCatalyst code version: H.0.
Copyright 1997-2005 Omniture, Inc. More info available at
http://www.omniture.com -->
<script language="JavaScript" src="//INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/s_code.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=" "
```

Redirects using getQueryParam

While the **getQueryParam** plug-in is an easy way to populate SiteCatalyst 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, which should be added to the "**doPlugins()**" function within the "*s_code_remote.js*" or "*s_code.js*" file as shown below.

```
H Code
/* Plugin Config */
s.usePlugins=true
function s_doPlugins(s) {
    /* Add calls to plugins here */
    // Copy original referrer into s.referrer if it exists
    var tempVar
    tempVar=s.getQueryParam('origref')
    if(tempVar)
        s.referrer=tempVar;
}
s.doPlugins=s_doPlugins
```

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 may need to check with your web operations group to identify the specific mechanisms that execute redirects on your website. The example below shows how one could pass along the original referrer and alias link information to the ultimate landing page using a client side redirect mechanism.

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, SiteCatalyst 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 SiteCatalyst 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.

G Code

```
<!-- SiteCatalyst code version: G.7.
Copyright 1997-2004 Omniture, Inc. More info available at
http://www.omniture.com --><script language="JavaScript"><!--
/* You may give each page an identifying name, server, and channel on
the next lines. */
var s_pageName=""
var s_server=""
var s_campaign=""
var s_referrer="http://www.google.com/search?hl=en&ie=UTF-
8&q=discount+airline+tickets"
// Setting the s_pageURL variable is optional.
var s_pageURL=http://www.flytohawaiiiforfree.com
```

H Code

```
<!-- SiteCatalyst code version: H.0.
Copyright 1997-2005 Omniture, Inc. More info available at
http://www.omniture.com -->
<script language="JavaScript"
src="http://INSERT-DOMAIN-AND-PATH-TO-CODE-HERE/s_code.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 SiteCatalyst 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 SiteCatalyst Debugger.

	URL or Query String Value	Value as Shown in the SiteCatalyst Debugger
Original Referrer	http://www.google.com/search%3Fhl%3Den%26ie%3DUTF826q%3Ddiscount%2Bairline%2Btickets	r=http://www.google.com/search?hl=en&ie=UTF-8&q=discount+airline+tickets
Page URL	http://www.flytohawaiiiforfree.com	g=http://www.flytohawaiiiforfree.com This value will appear in the SiteCatalyst Debugger if the <i>pageURL</i> variable is used.
Ultimate Landing Page URL	http://www.flywithus.com	This value will NOT appear in the SiteCatalyst 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/H.0-Pd-R/s61944015791667?[AQB]
ndh=1
t=11/5/2004 12:4:57 5 360
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 SiteCatalyst 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 the SiteCatalyst interface.

Report to Variable Mapping

The tables below display the report to variable mapping, or the SiteCatalyst 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 within SiteCatalyst:

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 - event20
...	...	Populate with event1 - event20
Custom Event 20	s.events	Populate with event1 - event20

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	

Campaigns		
Creative Elements	N/A	Defined in SiteCatalyst
Campaigns	N/A	Defined in SiteCatalyst

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

Technology		
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 SiteCatalyst
Time Spent on Site	N/A	Calculated by business rules in SiteCatalyst
Site Sections	s.channel	Same as Channel report in Traffic reports section

Customer Variables		
Customer eVar 1	s.eVar 1	
Customer eVar 2	s.eVar 2	
Customer eVar 3	s.eVar 3	
...		
Customer eVar 20	s.eVar 20	

Traffic Reports

The following table lists the **traffic** variables that are used to populate each report within SiteCatalyst:

Calculated Metrics		
N/A	N/A	N/A

Site Traffic		
Page Views	N/A	Calculated by business rules in SiteCatalyst
Hourly Unique Visitors	N/A	Calculated by business rules in SiteCatalyst
Daily Unique Visitors	N/A	Calculated by business rules in SiteCatalyst
Monthly Unique Visitors	N/A	Calculated by business rules in SiteCatalyst
Yearly Unique Visitors	N/A	Calculated by business rules in SiteCatalyst

Site Traffic		
Visits	N/A	Calculated by business rules in SiteCatalyst
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 SiteCatalyst
Daily Return Visits	N/A	Calculated by business rules in SiteCatalyst
Return Visits	N/A	Calculated by business rules in SiteCatalyst
Visit Numbers	N/A	Calculated by business rules in SiteCatalyst

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 SiteCatalyst
User Home Page	N/A	Automatically set by the .JS file
Key Visitors	N/A	Based on visitor's IP address
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 20	s.prop20	

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 SiteCatalyst:

Pages		
Page Summary	s.pageName (or other pathed variable)	Also depends on internal business rules

Pages		
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 SiteCatalyst reports.

Each variable is listed with the reports that use the variable listed next to it.

Variable	Reports Populated
s.pageName	Conversion Reports > Path Reports > Page Value Conversion Reports > Path Reports > Entry Page Conversion Reports > Path Reports > Original Entry Pages

Variable	Reports Populated
	<p>Traffic Reports > Site Traffic > Page Views</p> <p>Traffic Reports > Site Traffic > Hourly Unique Visitors</p> <p>Traffic Reports > Site Traffic > Daily Unique Visitors</p> <p>Traffic Reports > Site Traffic > Monthly Unique Visitors</p> <p>Traffic Reports > Site Traffic > Yearly Unique Visitors</p> <p>Traffic Reports > Visitor Profile > Pages Viewed by Key Visitors</p> <p>Traffic Reports > Segmentation > Most Popular Pages</p> <p>Path Reports > Pages > Page Summary</p> <p>Path Reports > Pages > Page Value</p> <p>Path Reports > Pages > Most Popular Pages</p> <p>Path Reports > Pages > Reloads</p> <p>Path Reports > Pages > Click to Page</p> <p>Path Reports > Pages > Time Spent on Page</p> <p>Path Reports > Entries & Exits > Entry Pages</p> <p>Path Reports > Entries & Exits > Exit Pages</p> <p>Path Reports > Entries & Exits > Exit Links</p> <p>Path Reports > Complete Paths > Full Paths</p> <p>Path Reports > Complete Paths > Longest Paths</p> <p>Path Reports > Complete Paths > Path Length</p> <p>Path Reports > Complete Paths > Time Spent per Visit</p> <p>Path Reports > Complete Paths > Single-page Visits</p> <p>Path Reports > Advanced Analysis > Previous Page</p> <p>Path Reports > Advanced Analysis > Next Page</p> <p>Path Reports > Advanced Analysis > Previous Page Flow</p> <p>Path Reports > Advanced Analysis > Next Page Flow</p> <p>Path Reports > Advanced Analysis > PathFinder</p> <p>Path Reports > Advanced Analysis > Fall-out</p> <p>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.</p>
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

Variable	Reports Populated
	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 Conversion Reports > Shopping Cart > Cart Views Conversion Reports > Shopping Cart > Cart Additions Conversion Reports > Shopping Cart > Cart Removals Conversion Reports > Shopping Cart > Checkouts Conversion Reports > Custom Events > Custom Event 1-20 Conversion Reports > Products > Conversion & Averages Conversion Reports > Products > Cross Sell Conversion Reports > Products > Categories Conversion Reports > Campaigns > Conversion & Averages 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.products	Conversion Reports > Purchases > Conversion & Averages Conversion Reports > Purchases > Revenue

Variable	Reports Populated
	<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> <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.purchaseID	<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 > 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.eVar1 - s.eVar20	Conversion Reports > Custom Variables > Customer eVar 1-20

Variable	Reports Populated
s.linkName	Traffic Reports > Custom Insight > Custom Links
s.hier1 - s.hier5	Traffic Reports > Hierarchies > Hierarchy 1-5

External Email Tracking

Companies use SiteCatalyst to determine the success of an e-mail campaign.

Companies often develop costly e-mail campaigns to market their products. The companies send bulk emails to customers with the expectation that some customers access the site and a conversion results. SiteCatalyst can report 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.
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 in SiteCatalyst. 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 ClientCare 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 SiteCatalyst JavaScript code in the JavaScript file, `s_code.js`, lets SiteCatalyst 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 ClientCare 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 in SiteCatalyst.

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 ClientCare 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.

Tracking External Emails

In addition to the external campaigns described in the previous section, SiteCatalyst can report campaign analysis data in several additional key metrics with configurations made by Adobe ClientCare.

Campaign Type	Description
Emails Sent	Displays the total number of emails sent by the company. This metric is provided by the email vendor.

Campaign Type	Description
Emails Delivered	Displays the total number of emails delivered to email servers. This metric is provided by the email vendor.
Emails Opened	Displays the number of times the visitors opened the email.

For more information on tracking email campaign analysis data, contact Adobe ClientCare. Tracking email campaign analysis data is considered on a case-by-case basis. This is done only through **Data Sources**.

Event Serialization

Event serialization is the process of removing duplicate events on each page view of the site with SiteCatalyst tags.

Event serialization is useful in the following instances:

- A page may be reloaded or refreshed and repeatedly send an event to SiteCatalyst. **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.

This document describes the process to implement **Event serialization** for **conversion** and **custom** events. To use **Event serialization**, you must first contact Adobe Live Support to have it enabled.

Default Behavior

The default behavior of SiteCatalyst code 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 within SiteCatalyst. 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.

- Use a unique identifier to let an event fire once per unique ID.
- Use SiteCatalyst's database 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 SiteCatalyst 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	SiteCatalyst Total Event1 Count
John	User views page for the first time.	event1:1000	1
John	User reloads the page (a form submit may fail, and cause the page to reload).	event1:1000	1
Stacy	User views page for the first time.	event1:1001	2
Stacy	User reloads the page (a form submit may fail, and cause the page to reload).	event1:1001	2
Jill	User views page for the first time, enters information correctly, and moves on to next page.	event1:1002	3
Jamie	User views page for the first time	event1	4
Jamie	User forgets to fill in the last name field on the form. The form is displayed again with the missing information highlighted.	event1	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).

For G Code, replace s. with s_.

```
s.events="[event]:[serial number]"
```

Samples:

```
s.events="event1:12341234"
s.events="event1:12341234,event2"
s.events="purchase,event1:12341234"
```

Serialization may be applied to both **custom events** (event1-event20), as well as pre-defined conversion events (**prodView**, **scView**, **scAdd**, **scRemove**, **scOpen**, **scCheckout**). Use the **s.purchaseID** variable to serialize the **purchase** event.

Event Serialization - Once per Visit

SiteCatalyst offers a feature to let an event only fire once per visit. Give the name of the report suite and event name to Adobe Live Support to enable the event.

Serializing eVars

The same functionality used to keep an event from being fired more than once per visit can be applied to **evar** instances. When enabled for an **evar**, **click-throughs** and **instances** are only counted once per visit for a specific **evar**. This is not counting once per value, but once per **evar**. This means that if an **evar** 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 **evar** 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

In most cases Adobe recommends using the **setValOnce** and **setOncePer** JavaScript plug-ins. Those plug-ins let a value be set only once, whereas the SiteCatalyst database lets the **eVar** be set only once.

Purchase Events

For the **purchase** event, SiteCatalyst 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.

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 from the Rapid Deployment Guide for SiteCatalyst (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).

Adobe Mobile Tracking

Mobile devices can be tracked and reported on through SiteCatalyst, Discover, DataWarehouse and SearchCenter. This document describes the tracking methodology and outlines differences between standard website and mobile tracking.

Tagging Mobile Pages

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.

The image beacon code can be generated from the **Administration Console Code Manager** by selecting the WAP code type. This code works for WAP and I-Mode networks. For information about how to create the image URL, refer to the Implementing without JavaScript white paper available in the SiteCatalyst Knowledge Base in the Help section. The guidelines below should be followed.

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.

How the Beacon Reaches Adobe

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 mobile networks are still in their infancy, and because screen size on 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.

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 new 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.

Reports for Mobile Devices

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

VISTA can be used to alter data collected from both Mobile and standard methods. All **Custom Insight (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.

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.

Time Zones, Cookies, Connection Types, 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.

Current Options to Improve Visitor Identification

The following are workaround options available to mitigate the inaccuracies found in the previous sections.

Define ASI Slots for Uniquely Identifiable Devices Only

If a client wishes to see a set of data representing only those devices that can be uniquely identified, an ASI segment containing only these hits can be defined. The reports run in this slot would be accurate, but would be subject to the limitations associated with a small data set (similar to sampling).

Dynamically Change the Format of the Hard-Coded Image Based on Whether or Not the Device Supports Cookies

The Adobe collection servers can set cookies on mobile devices if the device requests an image from the appropriate path (/b/ss/rsid/1). The '1' in the path forces Adobe servers to return a gif image and to attempt to set a persistent **visitor ID** cookie (s_vi).

Our standard wireless implementation specifies that '5' should be passed, which indicates that a wbmp should be returned and that our list of recognized wireless headers (not a cookie) should be used to identify the device.

These codes are mutually-exclusive, therefore always passing '5' does not leverage a cookie when it is supported. You can incorporate your own mechanism whereby they can 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 incremental number of devices supporting cookies.

Manage and Pass Unique Visitor IDs to Adobe

Should you have the ability to derive and manage the **visitor IDs** of their users, Adobe can provide you with three options to pass these IDs.

Via the Data Insertion API	This approach allows you to send an XML post containing the <visitorid/> XML element to Adobe collection servers from their servers.
Via a Query String Parameter on the Image Request	This capability was released in January 2007, and lets you pass the visitor ID to Adobe via the vid query string parameter on the hard-coded image request.
Via 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 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.

When viable, each of these approaches improves the accuracy issues associated with uniquely identifying a device/visitor.

Additional Information

Many older devices do not support wildcard SSL certificates, for example, devices running on Windows Mobile 5.0. If you are planning on implementing tracking on secure pages, and want to support older devices, you should consider a first-party collection domain implementation, since 2o7.net returns a wildcard certificate.

Report Suite IDs - Dynamic Accounts

The SiteCatalyst .JS file can be configured to automatically select a **Report Suite ID** (sometimes referred to as an account), based on the occurrence of a specific string or strings.

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 SiteCatalyst 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

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 ClientCare can provide help in troubleshooting.

Contact and Legal Information

Information to help you contact Adobe and to understand the legal issues concerning your use of this product and documentation.

Account Support

The Adobe support team is here to:

- Answer specific product questions
- Ensure that you can use the product to its maximum capacity
- Help resolve any technical difficulties you might have

Service and Billing Information

Depending on your service level, some options described in this manual might not be available to you. Because each account is unique, 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 Support Manager.

Feedback

We welcome any suggestions or feedback. Please send comments about the product to your Account Support Manager. For comments about this manual, use the **Feedback** button on each help topic.

Contact Information

Corporate address

Adobe Systems Inc.
3900 Adobe Way
Lehi, UT 84043

Phone: 1-408-916-9526

Fax: 1-385-345-0001

Toll Free: 1-800-497-0335 (support, billing, and sales)

Corporate URL: <http://helpx.adobe.com/contact.html>

Log-in URL: <http://my.omniture.com>

Legal Information

© 2013, Adobe
All rights reserved.
Published by Adobe Systems Inc.

[Terms of Use](#) | [Privacy Center](#)

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>.

01282013