Adobe® Marketing Cloud
Data Workbench Page Tagging
Contents

Data Workbench Page Tagging.................................................................................................................4
  What Kind of Data Can I Acquire?........................................................................................................4
  How Do I Acquire this Data..................................................................................................................5

Implementing Reference Page Tags........................................................................................................6
  Editing the Reference Page Tag Execution Script.................................................................................6
  Adding Reference Page Tag Execution Calls........................................................................................7
  Understanding the v1st Cookie.................................................................................................................8

Acquiring Baseline Measurements...........................................................................................................9
  Acquiring Page Request Data................................................................................................................9
    Understanding the Query String........................................................................................................10
    Filtering By Content Type..................................................................................................................11
    Performing Controlled Experiments.................................................................................................11
  Acquiring Embedded Object Requests (Page Tags)...............................................................................11
  Acquiring Measurement Data Through Cookies................................................................................13

Acquiring Extended Measurements.........................................................................................................15
  Acquiring Dynamic Page Names........................................................................................................15
  Acquiring Additional Information.......................................................................................................16
  Acquiring Document Objects.............................................................................................................16
  Measuring Advertisement Impression..................................................................................................17
  Tracking Link Clicks...........................................................................................................................18
  Tracking Exits to External Links..........................................................................................................19
  Acquiring Form Field Input...................................................................................................................20
    General Information........................................................................................................................20
    .ASP-Specific Information................................................................................................................21

Tracking Visitor Activity Within Flash Rich Media Content.....................................................................23

Last updated 1/31/2017

Data Workbench Page Tagging
Contents

P3P Considerations for Third-Party Page Tagging..............................................25
Page Tag Variable Mapping Worksheet..............................................................27
Data Workbench Page Tagging

Conceptual information about page tagging, including how to acquire data and the types of date you can acquire.

What Kind of Data Can I Acquire?

Sensor enables you to acquire web request data (event or log data) as well as extended measurement data. Extended measurement data is not available to your web servers as a part of their normal operation.

The following topics are described:

Web Request Data

Extended Measurement Data

Web Request Data

Sensor enables web request data (event or log data) to be acquired and transported automatically to a central location for storage and processing for analysis. Unless you specifically choose to filter out certain types of requests and not collect data about those request types, Sensor captures data about all GET requests made of the web servers on which it is installed.

Sensor automates this data acquisition process for all GET requests that are made on your servers and has significant business and technical benefits over alternative methods of acquiring website request data. These benefits include the following:

• Requests that are unnecessary for analysis and reporting can be filtered out before you incur costs for their acquisition, transportation, storage, and processing.
• Site administrators do not have to rotate log files in batch, either manually or via script.
• Sensor aggregates log files at a central location to allow easy access for processing.
• Sensor organizes and stores log files in a common data-preserving format, removing the need to preprocess them before they can be used for analysis and reporting purposes.
• Instances of certain content types can be included in the log files even though most requests for a certain content type are automatically filtered out.
• Sensor compresses log file entries, which requires significantly less storage space, reducing costs and allowing the data to be kept available for analysis for longer periods of time.
• Sensor's fault tolerant features allow system and network faults while still ensuring the delivery of the log data to a central repository.
• Sensor allows the implementation of controlled experiments with web content, processes, and marketing campaigns.
• Sensor time stamps log entries in 100ns units, allowing new types of analytic functionality.
• Sensor allows site owners to add data (measurements) to the log entries after initial implementation for consideration in analysis and reporting.

For more information about acquiring extended measurement data, see Acquiring Baseline Measurements.

Extended Measurement Data

Sensor also supports the use of page tags (or embedded object requests) to acquire measurement data that is not available to your web servers as a part of their normal operation. Page tags are commonly used to measure:

• The viewing of a logical page in a dynamic website.
• The viewing of content or ads on a third-party controlled website.
• The viewing of content that is served from a browser cache or CDN.
• Detailed information about a visitor's browser, including measurements such as page load time, screen resolution, what form fields the visitor has filled in, and so on.
• Other data that is not otherwise sent by browsers to your web servers.

Sensor collects any information placed in any GET request made to a web server that is running Sensor. Such requests may come from embedded object requests of any sort, either to simply measure that the request was made at a certain time by a certain browser or to pass other measurement data into the data collection stream so that it may be processed for analysis and reporting purposes.

Sensor provides the best of both client-side and server-side data acquisition worlds—it acquires your server-side web log data and collects client-side, third-party site, or cache-busting measurements taken by embedded object requests. In other words, Sensor acquires both the request data normally known to your web servers (server-side measurements) and any additional measurement data that you collect through the use of page tags (client-side measurements) that send their data to any web servers running Sensor. Such web servers can be dedicated to collecting client-side measurements but are not required to be.

For more information about acquiring extended measurement data, see Acquiring Extended Measurements.

How Do I Acquire this Data

You must install and run Sensor on each web server that serves the content for your site to collect all of the requests that are seen by those servers.

These requests make up 90% or more of the requests made to your site and 90% or more of the data that is needed for the complete analysis of your site’s traffic. PageTags should then be used to collect the remaining 10% or less of the traffic data that is not known to your web servers. The following, however, are valid configurations for the collection of web request data from your site, in order of preference, based on our operational experience:

1. Sensor is installed on each web server that you control and that supports your site. Content from third-party sites, content served from cache, and certain types of dynamic content should be tagged, and such page tags should send the data that they collect to a web server at your location that is running Sensor. You may add an additional web server if the level of page tag request traffic justifies such, or in special cases, dedicate a web server to collect these page tag requests.

2. Sensor is installed on two web servers, also referred to as data collection servers in this guide, at your location that are dedicated to collecting page tag request data from tagged pages. All content on your site is tagged and all page tags are directed to the two data collection servers.

3. Sensor’s data collection services are provided by an outsourcer that runs data collection servers to collect all of your web request data. In this case, all content on your site is tagged and the page tags send their data to the outsourced data collection servers.

For more information about Sensor, see the Data WorkbenchSensor Guide.
Implementing Reference Page Tags

Information about editing and adding Reference Page Tag Execution Scripts and Page Tag Execution Calls, and understanding the v1st Cookie.

Editing the Reference Page Tag Execution Script

The Reference Page Tag consists of a Page Tag Execution Script that resides on a web server, and when called results in the collection of all client-side data for the page requested by the site visitor.

You can modify the Reference Page Tag Execution Script to collect additional information that may be identified during requirements gathering meetings with the Adobe Consulting Services team. The Reference Page Tag Execution Script is relatively small in size to avoid large download additions to your web pages.

The following Reference Page Tag Execution Script code is provided to you in a file named zig.js:

```javascript
//REFERENCE PAGE TAG
// CONSTANTS
var ct = "<img src=";
var cd = "[PATH_TO_WEB_SERVER]"; //this should contain the domain of
//the web site that will host the
//page tag

var cu = "[PATH_TO_WEB_PAGE_TAG_CODE]/zag.gif?Log=1";
//this should contain the full path to
//the zag.gif file (excluding domain)
//and include the query string of log=1

var ce = ">

var c = {;
    c["sw"] = screen.width;
    c["sh"] = screen.height;
    c["cd"] = screen.colorDepth;
    var co = "";

    for ( cKey in c ) {
        co = co+"&"+cKey+"="+escape(c[cKey]);
    }
    document.write(ct,cd,cu,co,ce);

    var d = {;
        d["dt"] = document.title;
        d["dr"] = document.referrer;
        d["cb"] = new Date().getTime();
        var vo = "";

        if (typeof v != "undefined") {
            for ( vKey in v ) {
                vo = vo+"&"+vKey+"="+escape(v[vKey]);
            }
        }
        for ( dKey in d ) {
            vo = vo+"&"+dKey+"="+escape(d[dKey]);
        }
        document.write(ct,cd,cu,vo,ce);
    //END REFERENCE PAGE TAG
```

To facilitate data collection through the use of the Reference Page Tag, complete the following steps:

1. Create or place the 1 pixel by 1 pixel image file named zag.gif into a directory present on your web server.
2. Modify the cd variable to reference the appropriate domain of your website or Adobe managed services domain from which the zag.gif file is referenced. The reference to the file is created through the execution of the zig.js file functions. For example:

//www.mysite.com
3. Modify the cu variable to reference the appropriate path to the location of the zag.gif file. For example

```
/scripts
```

4. Ensure appropriate cache-control headers are established for the zag.gif and zig.js files.

## Adding Reference Page Tag Execution Calls

The **Reference Page Tag Execution Call** is inserted into web pages for which you want to collect measurement data.

It should be included in the body of the HTML document and can be placed within a global include footer if applicable. The **Reference Page Tag Execution Call** can be modified by your team to collect additional information that might be identified during requirements gathering meetings with the Adobe Consulting Services team.

To facilitate data collection through the use of the **Reference Page Tag**, complete the following steps:

1. Copy the following code into your HTML document body:

   ```html
   <!--//BEGIN REFERENCE PAGE TAG-->
   <script language="javascript">
   var vlc = "0" //Capture Link Click  1=TRUE, 0=FALSE
   var v = {};
   </script>
   <!--//MODIFY PATH TO ZIG.JS-->
   <script language="javascript" src="/path/to/zig.js" type="text/javascript"></script>
   <!--//END REFERENCE PAGE TAG-->
   <noscript>
   <img src="/path/to/zag.gif?Log=1&v_jd=1" border="0" width="1" height="1"/>
   </noscript>
   <!-- END REFERENCE PAGE TAG-->
   
2. Modify the path to the location of the zig.js and zag.gif files. For example:

   ```
   //www.mysite.com/scripts/zig.js
   //www.mysite.com/images/zag.gif
   ```

Please ensure that the appropriate HTTP Cache-Control headers have been set on your web server to ensure that the zig.js and zag.gif files are not cached by the browser. You can set the HTTP Cache-Control header information using one of two methods. The first method is to set an HTTP header via the web server. The second method is to set an HTTP header for each specific page or embedded object using script. With the scripting method, the web page must have been created using a programming language such as JSP or ASP. The page then is scripted so that it sends the appropriate header information. Two obvious drawbacks accompany this method: 1) all pages must be coded to send the header, and 2) the pages cannot be static HTML, which has some effect on web server performance.

Web sites running on Microsoft IIS can add the appropriate HTTP header through the Microsoft Management Console. Websites served from Netscape iPlanet Web Servers can accomplish this by editing the `obj.conf` file within the site’s configuration directory. The Apache Web Server provides webmasters the ability to customize HTTP headers using the included mod_headers module where AOLServer becomes customizable through the use of Tcl modules. Before implementing HTTP Cache-Control headers, you should refer to the documentation specific to your web server platform.
In general, the HTTP header should be structured as follows:

```plaintext
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
```

## Understanding the v1st Cookie

**Site** uses cookies to uniquely identify visitors to your website and track their behavior over time.

The first time that a particular browser (considered a visitor) makes a request of your website, **Sensor** works with your web server to set a persistent cookie, cs(cookie)(v1st), which is interpreted internally within the system as x-trackingid. This persistent cookie is set in addition to any other cookie that your site may otherwise set. This cookie optimizes your ability to track your visitors over multiple sessions, which enables many types of analysis that are otherwise impossible.

**Sensor** assigns a 64-bit numeric key in the cookie to identify new visitors that make a request of the site as a unique identifier. When the **Sensor** sees a request from a browser, it checks to see whether "cs(cookie)(v1st)”, a first-party persistent cookie set by **Sensor**, exists in the request data. If the cs(cookie)(v1st) is not present, **Sensor**, through your web server, tells the browser to set it. Unlike other solutions, **Sensor** is able to set this cookie on the visitor’s first request.

Below is the standard persistent cookie header sent to the browser on its first request of your site by your web server, at the direction of **Sensor**. The format can be defined at the time of configuration if a different name or expiration date is desired. For example:

```plaintext
Set-Cookie:v1st=3D80DCA944D60E16; path=/; expires=Wed, 19 Feb 2020 14:28:00 GMT
```

This cookie is set just once, on the very first request made to your site by that visitor. It then is collected from that visitor each time that browser makes a request (either page or embedded object request) of your site in the future. The cookie is very small in size to minimize the amount of bandwidth used to transmit it to your servers with each request from that browser to your site.

Accepting a persistent cookie is at the browser’s discretion. Most web users understand what cookies do and also recognize that first-party cookies provide a valuable benefit to them in allowing site content to be customized to their preferences. These first-party cookies are not blocked by the default security settings of the popular browsers. If a user does choose to block first-party cookies, their page view requests are still logged, but the measurement data from those requests cannot be reliably correlated to a particular visitor or their sessions on the site. Many sites, especially sophisticated dynamic sites, already use first-party cookies, which are in many cases necessary to enable web applications to operate for the visitor. A step back from a persistent cookie is a session cookie, which allows a series of requests to be knit together into a session, but does not allow inter-session visitor tracking. **Site** is capable of sessionizing visitor data based on session cookies or by IP number, but both methods significantly detract from the types and value of analysis that can be conducted with **Site** or any other web activity analysis and reporting system.
Acquiring Baseline Measurements

Conceptual information about acquiring page request data, embedded object requests, and measurement data through cookies.

Acquiring Page Request Data

Sensor acquires all measurement data that is carried on the page requests (GET requests) made to the web servers on which it has been installed.

Sensor acquires this measurement data through the web server’s application programming interface, directly from the instance or instances of the web server software running on your web server. Sensor does not access the web server generated log files. In fact, after Sensor and the data workbench server have been installed and tested, the web server’s native logging feature can be disabled without affecting data collection. In many cases, disabling the logging of files to the local disks of the web server machines themselves improves the page serving capacity of those web servers because of the relatively large amount of fixed disk I/O required to log this information to the local disk of the web server machine.

Sensor collects measurement and web request data directly from each web server process and virtual web server process (if applicable) and temporarily writes the data to a Queue File, a fault-tolerant memory queue with fixed disk backing, on the web server machine. The Sensor Transmitter service (or daemon depending upon the platform) retrieves data from the Queue File and then compresses and encrypts it before transmitting it to the data workbench server for long-term storage. With Sensor, data is accumulated on your web server machines in the Queue File only if you are having a network or other problem that prevents its transmission. The Queue File allows for the efficient local storage of hours to days of web request data to protect the data if a network or system fault does not allow the data to be transmitted to the data workbench server in real time.

Sensor collects measurement data from each physical and logical web server process, filters it by content type, compresses it, encrypts it, and streams it to the data workbench server.

The following table contains the fields of log information that are acquired by Sensor for each GET request that is not filtered out based on Sensor’s configuration file:

<table>
<thead>
<tr>
<th>W3C Name</th>
<th>Data Collected</th>
<th>Explanation</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-trackingid</td>
<td>Tracking Identifier (unique visitor)</td>
<td>Identifier read from a cookie placed in the user’s browser by Sensor on Visitor’s initial request</td>
<td>V1st=3C94007B4E01F9C2</td>
</tr>
<tr>
<td>Date Time</td>
<td>Timestamp</td>
<td>Time at which request was processed by server (at 100ns precision; accuracy depends on server environment and NTP)</td>
<td>2002-11-21 17:21:45.123</td>
</tr>
<tr>
<td>sc(content-Type)</td>
<td>Content Type</td>
<td>Type of object returned from server</td>
<td>text/html</td>
</tr>
<tr>
<td>sc-status</td>
<td>HTTP Response Status Code</td>
<td>Numerical code generated by the server that notes the status of the HTTP server’s response</td>
<td>404</td>
</tr>
</tbody>
</table>
Acquiring Baseline Measurements

<table>
<thead>
<tr>
<th>W3C Name</th>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs-uri-stem</td>
<td>URI Stem</td>
<td>The stem portion of the URI requested by the client</td>
<td>pagedir/page.asp</td>
</tr>
<tr>
<td>c-ip</td>
<td>Client IP</td>
<td>IP Address of the requesting client</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>s-dns</td>
<td>Server Domain Name</td>
<td>Domain name of the web server processing the request</td>
<td><a href="http://www.domain.com">www.domain.com</a></td>
</tr>
<tr>
<td>cs(referer)</td>
<td>Referring URL</td>
<td>Contents of the HTTP referrer field sent by the client</td>
<td><a href="http://www.referringsite.com">http://www.referringsite.com</a></td>
</tr>
<tr>
<td>cs(user-agent)</td>
<td>User Agent</td>
<td>Device used to make a request to the HTTP server</td>
<td>Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1)</td>
</tr>
<tr>
<td>cs(cookie)</td>
<td>Client Cookies from Domain</td>
<td>Contents of all of the user's cookies for the site</td>
<td>KL_TC1 1038058778312</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KL972x10380587783122282052</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KL_PVKL972 0</td>
</tr>
<tr>
<td>cs-uri-query</td>
<td>Query String</td>
<td>The query string portion, if any, of the URI requested by the client</td>
<td>PAGENAME=dynamic1&amp;link=3001</td>
</tr>
</tbody>
</table>

Understanding the Query String

The query string (cs-uri-query) is often used by web applications and site developers to pass information from page to page because of the stateless nature of HTTP.

In many cases, information may be passed in the query string when it is acquired by Sensor at the web server. Such information can be used by Site to illuminate the true structure of the site, and the path of visitors through it, as well as other information.

In some dynamic websites, name=value pairs (variables) in the query string are important for determining the actual page being requested by a visitor. In such cases, URLs may be structured in the following or similar manner:

```
http://www.myserver.com/pageserved.asp?PAGENAME=HOME
```

In this example, PAGENAME is actually the indicator of what page will be served to the requester of this URL. Many web log analysis tools and services limit a site operator’s ability to define what a page is in their site based on what query string variables occur in the query strings of the site’s URLs. The data workbench server and data workbench can be configured to use such query names to define unique pages. This is important because many systems would interpret the following URLs as the same page, but Site does not.

```
http://www.myserver.com/pageserved.asp?PAGENAME=HOME
http://www.myserver.com/pageserved.asp?PAGENAME=HOME2
```

Similarly, site developers and applications often add many query string variables into a site’s URLs that have nothing to do with identifying the actual page that is being requested. Examples are shown below:

```
http://www.myserver.com/pageserved.asp?PAGENAME=HOME&CAMPAIGN=10001
http://www.myserver.com/pageserved.asp?PAGENAME=HOME&CAMPAIGN=10002
http://www.myserver.com/pageserved.asp?PAGENAME=HOME&CAMPAIGN=10003
```

In this example, the query string variable CAMPAIGN= has been added to the URL. This CAMPAIGN variable is being used to indicate which marketing campaign caused a visitor to select this URL. Site can be configured to use...
this CAMPAIGN information, yet separate it from the definition of what page a visitor viewed so that in your list of pages for reporting and analysis purposes you would simply see the following:

http://www.myserver.com/pageserved.asp?PAGENAME=HOME

**Filtering By Content Type**

The purpose of Sensor's content-type filtering capability is to eliminate the need to store and process information that is not useful for analysis purposes.

Much of the request data that is available through a web server's API is not useful in business analysis. Storage and processing are expensive and Sensor's content-type filtering allows you to avoid unnecessary storage and processing.

To maximize the web log data processing performance and reduce the amount of measurement data that must be stored, Site acquires measurement data (request data, log entries, log data, and so on) for all web content-type requests, except for specifically listed content types (such as cascading style sheets, image requests, and so forth), which are filtered out before they are transmitted to the data workbench server by Sensor. This filtering can be disabled for an entire web server, and it also can be overridden for a particular content object by adding the name-value pair “Log=1” to the query string of a particular embedded object (for example, http://www.mysite.com/advertisement.gif?Log=1).

**Performing Controlled Experiments**

Sensor enables controlled website experimentation, allowing you to run controlled experiments on random subsets of your visitor population with minimal disruption to your ongoing operations.

Controlled experiments allow experimentation with hypotheses for improving website financial performance, improving customer experience, or influencing customer behavior. Marketing programs, content personalization initiatives, process flows, and creative or content concept hypotheses can be tested on a controlled percentage of randomly selected site visitors and then analyzed in data workbench to determine the merit of the hypotheses.

**Acquiring Embedded Object Requests (Page Tags)**

After the HTML of a page is requested by a browser, the browser requests the embedded objects referenced in the HTML of that page from a web server to fill in the page shown by the browser.

Such embedded object requests are most commonly requests for image files or JavaScript files, though there are hundreds or maybe thousands of types of embedded objects used on the Internet today. Many of these embedded object requests are not generally useful in analyzing or reporting on the business activity of an Internet site; many such requests are therefore not desirable for acquisition unless they have a specific business purpose, such as presenting an advertisement or taking another measurement of site activity.

For example, an image may be an advertisement, and you may want to know that the advertisement was impressed upon a visitor. A JavaScript snippet may be in use to take a measurement that the particular browser has a certain characteristic and pass it back to a Sensor for acquisition. Each page on a site may have 10 or 100 embedded object requests in it. If a site stores the log information for each of these requests, the amount of data storage needed to keep the log data available for future analysis is multiplied by the number of embedded object requests for each page requested. For this reason, Site lets you keep the requests that are important for analysis and discard others before you incur unnecessary storage costs.
By using the override feature provided in the content-type filtering capabilities of Sensor (appending “Log=1” to the query string of an embedded object request URL), that particular embedded object request and the related measurement data can be acquired without requiring the site manager to store all requests of that type (for example, all <image> requests).

Sensor collects the measurement data in the following table for each embedded object request made of the web server, assuming that Sensor is not configured to filter it out or that the filter has been overridden. The collected information is related to the visitor and session and subsequent sessions through the x-trackingid or cs(cookie) log field entries.

<table>
<thead>
<tr>
<th>W3C Name</th>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-trackingid</td>
<td>Tracking Identifier (unique visitor)</td>
<td>Identifier read from a cookie placed in the user’s browser by Sensor on initial request</td>
<td>V1st=3C94007B4E01F9C2</td>
</tr>
<tr>
<td>Date</td>
<td>Time</td>
<td>Time at which request was processed by server (at 100ns precision; accuracy depends on server environment and NTP)</td>
<td>2002-11-21 17:21:45.123</td>
</tr>
<tr>
<td>sc(content-Type)</td>
<td>Content Type</td>
<td>Type of object returned from server</td>
<td>text/html</td>
</tr>
<tr>
<td>sc-status</td>
<td>HTTP Response Status Code</td>
<td>Numerical code generated by the server that notes the status of the HTTP server's response</td>
<td>200</td>
</tr>
<tr>
<td>cs-uri-stem</td>
<td>URI Stem</td>
<td>The “stem” portion of the URI requested by the client</td>
<td>pagedir//page.asp</td>
</tr>
<tr>
<td>c-ip</td>
<td>Client IP</td>
<td>IP Address of the requesting client</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>s-dns</td>
<td>Server Domain Name</td>
<td>Domain name of the web server processing the request</td>
<td><a href="http://www.domain.com">www.domain.com</a></td>
</tr>
<tr>
<td>cs(referrer)</td>
<td>Referring URL</td>
<td>Contents of the HTTP referrer field sent by the client</td>
<td><a href="http://www.referringsite.com">http://www.referringsite.com</a></td>
</tr>
<tr>
<td>cs(user-agent)</td>
<td>User Agent</td>
<td>Device used to make a request to the HTTP server</td>
<td>Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.1)</td>
</tr>
</tbody>
</table>
| cs(cookie)   | Client Cookies from Domain      | Contents of all of the user’s cookies for the site                           | KL_TC1 1038058778312
KL972 x1038058778312282052
KL_PVKL972 0                                      |
| cs-uri-query| Query String                    | The query string portion, if any, of the URI requested by the client         | PAGENAME=dynamic1&link=3001                  |
Acquiring Measurement Data Through Cookies

As a part of the Baseline Measurement data collected, Sensor collects the domain cookies sent from a visitor’s machine when making a request from your web server. This includes both persistent and session cookies that your website sets when a visitor interacts with your system. In most cases, websites set persistent cookies to identify visitors or capture user input for use within subsequent visitor sessions. Any information written to and stored within persistent cookies can be captured and used alongside all other measurement data within the data workbench server.

An example of such a persistent cookie could involve a customer identifier in the form of a numeric key present within a domain specific cookie residing on the visitor’s machine. In addition to identifying the user as a return visitor, the persistent cookie could also be used to further identify the visitor as a returning customer or to tie the visitor to information contained within a customer database to allow offline customer demographic information to be displayed within Site and used for interactive analysis.

Session cookies can be a good mechanism to collect user input through form fields or other dynamic interactive elements within your website. In the case of a website implementing forms to capture user-specific input data, the information remains in the session cookie only for as long as the session is active. When a user leaves your website or subsequently ends a session, the information is no longer stored on the user’s computer. However, the information entered is captured by Sensor and made available as measurement data within Site.

Following is an example of using a session cookie to capture a single form variable entered by a visitor.

```
<html>
<head>
<title>Cookie Collection </title>
<script language="JavaScript">
function AppendFormValues()
{

var item = document.testform.elements[i];
var formitem = "v_"+i;
var formvalue = item.value;
cookie += formitem + "=" + formvalue + ";"

document.cookie = cookie;

testform.submit();
}
</script>
</head>
<body>
<form name="testform" method="post" action="nextpage.asp">
<input type="text" size=15 name="name"><br />
<br />
<a href="javascript:AppendFormValues();">Click Here To </a>
<br />
<br />
<br />
</body>
</html>
```

In this example, a function is called to set a session cookie on the visitor’s machine with the name of the field and the value entered into the form field. As the form is submitted, and the subsequent web page is requested, the session cookie set is passed to the web server and collected by Sensor. The following data is therefore available within the data workbench server for use in data analysis:
Session cookies may also be utilized to iteratively capture form fields or a multitude of embedded JavaScript variables present within an HTML page. In the following example, JavaScript is used to recursively capture any form field present within an HTML file and set a session cookie with the appropriate name=value pairs.

```javascript
<script language="JavaScript">

function AppendFormValues()
{
    var cookie="formcookie=";
    for (i=0; i<document.testform.length; i++){
        if (document.testform.elements[i].type == "radio") {
            if (document.testform.elements[i].checked){
                var formitem = "v_"+i;
                var formvalue = item.value;
                cookie += formitem + "=" + formvalue + "&";
            }
        } else if (document.testform.elements[i].type == "select") {
            var item = document.testform.elements[i];
            var formitem = "v_"+i;
            var formvalue = document.testform.elements[i].options[optionindex].value;
            cookie += formitem + "=" + formvalue + "&";
        } else{
            var item = document.testform.elements[i];
            var formitem = "v_"+i;
            var formvalue = item.value;
            cookie += formitem + "=" + formvalue + "&";
        }
    }
    document.cookie = cookie;
    document.testform.submit();
}
</script>
```

In this example, a session cookie is set on the visitor’s machine with the name and value of every form field that exists within the form. This includes input fields, check boxes, radio buttons, select boxes, and text areas. As you may notice in this example, because the number of form fields is unknown, it is necessary to capture all form name and field values as a single string, delimited by an ampersand. This step must be taken because of a limit on the number of cookies a user may have on his or her computer at one point in time. Microsoft Internet Explorer allows only twenty (20) session cookies to be present before it begins dropping the oldest.
Acquiring Extended Measurements

Conceptual information on acquiring extended measurements, and tracking link clicks and exits to external links.

Acquiring Dynamic Page Names

For some sites, it is necessary to use embedded object requests to pass information to the web server so that details about what page was actually served may be acquired by Sensor and used for reporting and analysis.

This might be required if the page's URL, as seen by the web server, is not indicative of the page content that is shown to the browser. This case often results from the use of personalization or dynamic content management systems in which the actual content served in a page is determined on the fly by the URL, the cookie, related data, and application logic.

The implementation of an embedded object to gather additional measurements should have minimal impact on your overall site performance. Adobe suggests that you embed a JavaScript file as the object used to collect the extended attributes. (Note that a JavaScript file can be embedded without any potential impact to the layout and presentation of your web page as may result with the use of an embedded image.) To accurately capture the information passed within the embedded object, Adobe also suggests that a common name be used. For naming purposes, Adobe refers to this object as zig.js. The zig.js file should be created within the appropriate directory on a web server on which Sensor is installed. This file needs to exist so that the request does not return a 404 error code. The contents of the file itself are not important. You should use a blank file named zig.js to minimize the amount of network traffic incurred as a result of the request.

For Sensor to collect a usable name for the page that was actually served, a few lines of JavaScript code must be added to the dynamic pages that you want to track or to which you want to add a unique page name. This code embeds a snippet of JavaScript in the page, which causes a tertiary embedded object request to be made to the web server as the page is loading. That request sends details about the specific page that was served back to the web server. The name of the page that was actually served is carried back to the web server as a variable in the query string of the embedded object (in this case JavaScript) request.

In general, the object request embedded in each such HTML page should look like the following:

```html
<!-- BEGIN REFERENCE PAGE TAG-->
<script language="javascript">
  var vlc = "0" //Capture Link Click  1=TRUE, 0=FALSE
  var v = {};
  v["_pn"] = "Application Form";
</script>

<script language="javascript" src="http://www.myserver.com/path/to/zig.js" type="text/javascript"></script>

<noscript>
  <img src="/path/to/zag.gif?Log=1&v_jd=1" border="0" width="1" height="1"/>
</noscript>

<!-- END REFERENCE PAGE TAG-->
```

Log=1 ensures that Sensor logs the request in spite of the Sensor content type filtering rules to the contrary, such as the filtering out of JavaScript and image requests before they are stored. The declared v_pn variable identifies the name of the actual page content being served so that Site knows the name of the page the visitor actually viewed. The v_pn value could be established manually or by other script or application code.

After the value is collected, you can configure the data workbench server to use the contents of the query string variable (name=value pair, for example, v_pn=Application Form) appended to the zag.gif URI (for example,
http://www.mysite.com/pageserved.asp?v_pn=Application%20Form), as an augmentation of the zag.gif URI. In addition to the baseline measurements acquired with every HTTP request, an extended measurement would be acquired with this request.

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_pn=</td>
<td>Value associated with the v_pn query string variable</td>
<td>v_pn=Application Form</td>
</tr>
</tbody>
</table>

**Acquiring Additional Information**

Query string variables may be added to a JavaScript request to collect additional measurements when a request is made.

These variables may be added manually or by script in the page itself.

Additional information that can be acquired from a page may be added to the embedded object via script using the following code as an example:

```javascript
<!-- BEGIN REFERENCE PAGE TAG-->
<script language="javascript">
    var vlc = "0" //Capture Link Click 1=TRUE, 0=FALSE
    var v = {};
    v["_pn"] = "Application Form";
    v["_1"] = "99.99";
    v["_2"] = "visa";
</script>
<script language="javascript" src="http://www.myserver.com/path/to/zig.js" type="text/javascript"></script>
<noscript>
    <img src="/path/to/zag.gif?Log=1&v_jd=1" border="0" width="1" height="1"/>
</noscript>
<!-- END REFERENCE PAGE TAG-->
```

In this example, the script variables for v_1 and v_2 can be derived from another function within your web page. The variables have been inserted as examples. In addition to the baseline measurements acquired with each request, the following extended measurements would be acquired with the request of the URL above:

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_pn=</td>
<td>Value associated with the v_pn query string variable</td>
<td>v_pn=Application Form</td>
</tr>
<tr>
<td>v_1=</td>
<td>Value associated with the v_1 query string variable</td>
<td>v_1=99.99</td>
</tr>
<tr>
<td>v_2=</td>
<td>Value associated with the v_2 query string variable</td>
<td>v_2=visa</td>
</tr>
</tbody>
</table>

**Acquiring Document Objects**

Using the JavaScript Document Object Model, additional scripting methods can be employed to augment the request for the zig.js file.
Information such as the value of META tags, ID values of DIV tags, and so forth, can be referenced by name and collected as variables for use in analysis. For example, to dynamically capture the information contained within the META element of the HTML document, you can use the following JavaScript syntax:

```javascript
<!-- BEGIN REFERENCE PAGE TAG-->
<script language="javascript">
    var m0 = document.getElementsByTagName('META')[0]; //define the first instance of META
    var metacontent = m0.getAttribute('content'); //get the 'content' value of META
    var vlc = "0" //Capture Link Click  1=TRUE, 0=FALSE
    var v = {};
    v["_1"] = metacontent;
</script>

<script language="javascript" src="http://www.myserver.com/path/to/zig.js" type="text/javascript"></script>

<noscript>
    <img src="/path/to/zag.gif?Log=1&v_jd=1" border="0" width="1" height="1"/>
</noscript>
<!-- END REFERENCE PAGE TAG-->
```

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_1=</td>
<td>Value associated with the METAVALUE query string variable. This value represents the data within the META element of the HTML document.</td>
<td>v_1=This page serves content related to the order thank you page.</td>
</tr>
</tbody>
</table>

After the data is collected, you can configure the data workbench server to process this measurement data for the purposes of analysis and reporting.

### Measuring Advertisement Impression

Marketing your website may involve the placement of advertisements in the form of image or other rich media files (served from your web server) on third-party websites.

In such cases, you might want to measure both the impression of the advertisement on a browser and the subsequent click-through, if one occurs, to the advertisement’s target URL on your website.

For advertisements in the form of images, appending `Log=1` to the query string results in the image request, and thus the advertisement impression, being captured by `Sensor` for analysis purposes.

```html
<!--REFERENCE IMPRESSION TAG-->
<!--END REFERENCE IMPRESSION TAG-->
```

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_ic=</td>
<td>Value denoting the Impression Campaign</td>
<td>v_ic=&quot;CAMPAIGN1&quot;</td>
</tr>
<tr>
<td>v_ica=</td>
<td>Value denoting the Impression Campaign Asset</td>
<td>v_ica=&quot;72890ab&quot;</td>
</tr>
<tr>
<td>v_icr=</td>
<td>Value denoting the Impression Campaign Referrer</td>
<td>v_icr=&quot;<a href="http://money.cnn.com/markets/">http://money.cnn.com/markets/</a>&quot;</td>
</tr>
</tbody>
</table>
In addition to appending \texttt{Log=1} to the image request, an identifier should be added to the URL leading from the advertisement to the target page within your website to track the advertisement that led to the click-through and to track the click-through back to the particular campaign for that advertisement.

\begin{verbatim}
\textbf{Example Explanation Data Collected}\\
v_c="CAMPAIGN1" Value denoting the Click-through Campaign
v_ca="72890ab" Value denoting the Click-through Campaign Asset
v_cr="http://money.cnn.com/markets/" Value denoting the Click-through Campaign Referrer
\end{verbatim}

\textbf{Tracking Link Clicks}

Steps used to facilitate the collection of Link Clicks through the use of the Reference Page Tag.

Through the deployment of the \textbf{Reference Page Tag}, it is possible to collect measurement data denoting the links (or \texttt{href} values) that visitors click while visiting particular pages. Typically, this collection does not involve the implementation of additional link identifiers into your HTML pages.

To facilitate collection of Link Clicks through the use of the \textbf{Reference Page Tag}, complete the following steps:

1. Copy the following code into the existing file named \texttt{zig.js}:

```javascript
//REFERENCE LINK AND FORM CLICK PAGE TAG
//INITIATE FUNCTIONS ONLOAD
function addEvent(obj, evType, fn){
  if (obj.addEventListener){
    obj.addEventListener(evType, fn, false);
    return true;
  } else if (obj.attachEvent){
    var r = obj.attachEvent("on"+evType, fn);
    return r;
  } else {
    return false;
  }
}
addEvent(window, 'load', startCapture);
addEvent(window, 'load', startCapture);
function startCapture(){
  //TO CAPTURE LINK CLICKS
  if (vlc == "1"){captureLink();}
  //TO CAPTURE FORM FIELD CLICKS
  if (vfc == "1"){captureForm();}
}
//BEGIN LINK CAPTURE PAGE TAG
function captureLink(){
  if (document.links[0]){  
    if (document.links){
      var links = document.links, link, k=0;
      while(link=links[k++]) {
        link.onclick = captureLinkName;
      }
    }
  }
}
```
2. Create or place the 1 pixel by 1 pixel image file named `zag2.gif` into a directory present on your web server.

3. Modify the `lc.src` variable to reference the appropriate domain of your website from which the `zag2.gif` file is referenced.

4. Ensure appropriate cache-control headers are established for the `zag.gif` and `zig.js` files.

5. Within the HTML files you desire to collect link click values from, the **Reference Page Tag Execution Call** must be modified to inform the **Page Tag Execution Script** to capture link clicks for that page. To do so, change the `vlc` variable value to “1” as highlighted in the following code example:

```html
<!-- BEGIN REFERENCE PAGE TAG-->
<script language="javascript">
    var vlc = "1" //Capture Link Click  1=TRUE, 0=FALSE
    var vfc = "0"; //Capture Form Field Click  1=TRUE, 0=FALSE
    var v = {};
</script>
<script language="javascript" src="http://www.myserver.com/path/to/zig.js"
type="text/javascript"></script>
<noscript>
    <img src="/path/to/zag.gif?Log=1&v_jd=1" border="0" width="1" height="1"/>
</noscript>
<!-- END REFERENCE PAGE TAG-->
```

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>v_ln</code></td>
<td>Value denoting the Impression Campaign</td>
<td><code>v_ln=&quot;About%20Us&quot;</code></td>
</tr>
</tbody>
</table>

### Tracking Exits to External Links

Capturing activity across third-party website links to enable Exit Target analysis.

Web pages can contain links to third-party websites, and activity across those links can be captured to enable Exit Target analysis, especially in the case that the third-party site is responsible for paying referral fees when such referrals are received. Because the click event is written to the log files of the third-party system by default,
modifications need to be made to the link for the click event to be captured locally. The third-party link present within your website must be modified as follows:

```html

The referenced PageExit.htm file must be created and should be structured to contain the following script:

```html
<html>
<head>
<script>
function getExitURLQuery(variable) {
    var query = window.location.search.substring(1);
    var vars = query.split("&");
    for (var i=0; i<vars.length; i++) {
        var pair = vars[i].split("=");
        if (pair[0] == variable) {
            return pair[1];
        }
    }
}
</script>
<script>
location.replace(getExitURLQuery("v_eurl"));
</script>
</head>
</html>
```

By making the request for the PageExit.htm file, the v_eurl value is collected for analysis purposes. Additionally, when PageExit.htm is loaded, it immediately redirects to the specified v_eurl target location.

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_eurl</td>
<td>Value associated with the v_eurl query string variable. This value represents the target URL of the link present within the HTML page.</td>
<td>v_eurl=www.othersite.com</td>
</tr>
</tbody>
</table>

**Acquiring Form Field Input**

Information on acquiring form field input, including general information and .asp-specific information.

**General Information**

Values entered into a form in a web page can be collected and appended in the query string of the subsequently requested page (on form submission) through the use of JavaScript.

This is shown in the following example. Include this JavaScript after any form validation scripts used in your HTML pages.

```html
<html>
<head>
</head>
<script language="JavaScript">
function AppendFormValues()
```
This example appends the values entered into the form by the browser user to the subsequent “thankyou.asp” page indicated in the FORM Action value as follows:

http://www.myserver.com/thankyou.asp?v_1=John Smith&v_2=Los Angeles&v_3=California&v_4=90210

The following extended measurements would be acquired with this request in addition to the baseline measurements collected by Sensor:

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_1</td>
<td>Value associated with the NAME form field</td>
<td>v_1=John Smith</td>
</tr>
<tr>
<td>v_2</td>
<td>Value associated with the CITY form field</td>
<td>v_2=Los Angeles</td>
</tr>
<tr>
<td>v_3</td>
<td>Value associated with the STATE form field</td>
<td>v_3=California</td>
</tr>
<tr>
<td>v_4</td>
<td>Value associated with the ZIP form field</td>
<td>v_4=90210</td>
</tr>
</tbody>
</table>

**ASP-Specific Information**

Web pages are often structured using ASP (Active Server Pages) programming language.

ASP is a Microsoft technology that runs within IIS (Internet Information Services). When a browser requests an ASP file, IIS passes the request to the ASP engine. The ASP engine reads the ASP file, line by line, and executes the scripts in the file. Finally, the ASP file is returned to the browser as plain HTML. ASP provides RESPOND or REQUEST objects which, in addition to other uses, allow the response or request of user queries or data submitted from HTML forms.

In certain cases, you may not want to append the values entered into forms to the URL that is displayed within the Address bar of a user's browser or that is viewable within the HTML code itself. Simple server-side JavaScript allows you to append form field names and their respective values to the log file without making them available within the user's browser or embedding them into the HTML file. To capture the actual form values entered into particular forms within your website, a few lines of code must be added to append the form values to the log request.
Within the processing page of a form, include the following code to append the entered form values to the request data (in addition to writing the submitted form values to an external database or other location):

```javascript
var sName= Request.Form("Name");
var sCity= Request.Form("City");
var sState= Request.Form("State");
var sZip= Request.Form("Zip");

Response.AppendToLog("&v_1=" + sName);
Response.AppendToLog("&v_2=" + sCity);
Response.AppendToLog("&v_3=" + sState);
Response.AppendToLog("&v_4=" + sZip);
```

This process would append the form values as defined to the request data for the Form Processing page. Within the log data, the appended values would be available as query strings of the Form Processing page as illustrated below. For example, v_1, v_2, v_3 and v_4 would now be query strings containing the data entered into the appropriate form fields. The syntax described in the example above can be duplicated for any additional form fields and values that you want to capture.

http://www.myserver.com/path/to/formprocessingpage.asp?v_1=John+Smith&v_2=Los+Angeles&v_3=California&v_4=90210

If you want every form field and value to be captured and available for analysis, you can use the following syntax:

```javascript
var formvalues = Response.Form;
Response.AppendToLog(formvalues);
```

This example would take all form fields present within the HTML along with their respective values and append them as query strings to the log entry for the Form Processing page. It should be noted that this would include any hidden fields present within the form.

The log data would be augmented as detailed in the following table:

<table>
<thead>
<tr>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_1</td>
<td>Value associated with the NAME query string</td>
<td>v_1=John Smith</td>
</tr>
<tr>
<td>v_2</td>
<td>Value associated with the CITY query string</td>
<td>v_2=Los Angeles</td>
</tr>
<tr>
<td>v_3</td>
<td>Value associated with the STATE query string</td>
<td>v_3=California</td>
</tr>
<tr>
<td>v_4</td>
<td>Value associated with the ZIP query string</td>
<td>v_4=90210</td>
</tr>
</tbody>
</table>
Tracking Visitor Activity Within Flash Rich Media Content

Websites architected using Flash require special attention with respect to the capture of visitor actions performed within the rich media content.

Using Flash ActionScript, you can make simple changes to your existing Flash movies to allow the tracking of all visitor interactions with the movie, such as button clicks or mouse movements.

To facilitate Visitor activity tracking within your Flash movie, please follow the steps listed below:

1. Add the following ActionScript code to your movie. This code represents a function that can be called by events within the Flash movie that you want to track.

```actionscript
// FLASH TAG CODE BEGIN
var FLASHTAGURI = "[PATH_TO_WEB_SERVER]/flashtag.txt";
function tag(PAGENAME,VARIABLES) {
loadVariablesNum(FLASHTAGURI+"?"+"PAGENAME="+PAGENAME+"&"+VARIABLES,0);
}
// FLASH TAG CODE END
```

2. Create a blank file named `flashtag.txt` and place the file on your web servers.

3. Within the function in Step 1, replace the `[PATH_TO_WEB_SERVER]` placeholder with the fully qualified or relative path to the location of the `flashtag.txt` file. For example:

```actionscript
var FLASHTAGURI = http://www.mysite.com/flashtag/flashtag.txt";
```

4. Add the following ActionScript code to all events to be tracked. This code represents a function call used to capture data about the event:

```actionscript
on(release) {tag("/[PUT_PAGE_NAME_HERE]","[PUT_ADDITIONAL_VAR_HERE]");}
```

This example illustrates the use of the on(release) event; however, the tag() function may be referenced through any event that you may want to track, such as an on(press), on(rollover), on(rollout), or on(keypress) event.

The `[PUT_PAGE_NAME_HERE]` placeholder should be replaced with a string that represents the name of the page or event that you are tracking. The `[PUT_PAGE_NAME_HERE]` variable can be modified either manually or through variable reference to denote a unique name for the page or event within the Flash application. The value replacing the `[PUT_PAGE_NAME_HERE]` placeholder may consist of a simple name or may be structured to represent a hierarchical structure similar to a full URI. For example:

```actionscript
on(release) {tag("/about_us/index.swf","[PUT_ADDITIONAL_VAR_HERE]"});
```

Adobe recommends that, prior to code deployment, you compile a written specification for page names and event names to facilitate the alignment of business requirements and development tasks and reduce the potential for additional development cycles.

5. If desired, additional variables may be collected and associated with pages or events in the Flash movie. To do so, replace the `[PUT_ADDITIONAL_VAR_HERE]` placeholder with a set of name=value pairs separated by an ampersand (&). For example:

```actionscript
on(release) {tag("/about_us/index.swf"," var1=value1&var2=value2");}
```

The variables can be modified either manually or through variable reference to denote additional attributes to be collected and associated with the page or event. If there are no applicable additional variables to collect, remove `[PUT_ADDITIONAL_VAR_HERE]`.

Your setup of visitor tracking within Flash rich media content is now complete. When the event is invoked, the tag (PAGENAME, VARIABLES) function will be called, resulting in an HTTP request being made for the following
file. This function will be called in addition to other functions that may be triggered as defined within your Flash movie:

http://www.mysite.com/flashtag/flashtag.txt?PAGENAME=/about_us/index.swf&var1=value1&var2=value2

The HTTP request resulting from the Flash Tag ActionScript function results in the following information being collected with respect to each event within the Flash movie. The last row in the table (W3C Name cs-uri-query) represents the information collected for the additional variables specified in your function call.

<table>
<thead>
<tr>
<th>W3C Name</th>
<th>Data Collected</th>
<th>Explanation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-trackingid</td>
<td>Identifier read from a cookie placed in the user's browser by Sensor on Visitor's initial request</td>
<td>v1st=3C94007B4E01F9C2</td>
<td></td>
</tr>
<tr>
<td>Date Time</td>
<td>Timestamp</td>
<td>Time at which request was processed by server (at 100ns precision; accuracy depends on server environment and NTP)</td>
<td>2002-11-21 17:21:45.123</td>
</tr>
<tr>
<td>sc(content-Type)</td>
<td>Content Type</td>
<td>Type of object returned from server</td>
<td>Text/html</td>
</tr>
<tr>
<td>sc-status</td>
<td>HTTP Response Status Code</td>
<td>Numerical code generated by the server that notes the status of the HTTP server's response</td>
<td>200</td>
</tr>
<tr>
<td>cs-uri-stem</td>
<td>URI Stem</td>
<td>The stem portion of the URI requested by the client</td>
<td>/flashtag/flashtag.txt</td>
</tr>
<tr>
<td>c-ip</td>
<td>Client IP</td>
<td>IP Address of the requesting client</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>s-dns</td>
<td>Server Domain Name</td>
<td>Domain name of the web server processing the request</td>
<td><a href="http://www.mysite.com">www.mysite.com</a></td>
</tr>
<tr>
<td>cs(referrer)</td>
<td>Referring URL</td>
<td>Contents of the HTTP referrer field sent by the client</td>
<td></td>
</tr>
<tr>
<td>cs(user-agent)</td>
<td>User Agent</td>
<td>Device used to make a request to the HTTP server</td>
<td>Mozilla/4.0+(compatible;+MSIE+6.0; +Windows+NT+5.1)</td>
</tr>
<tr>
<td>cs(cookie)</td>
<td>Client Cookies from Domain</td>
<td>Contents of all of the user's cookies for the site</td>
<td>KL_TC1 1038058778312</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KL972x1038058778312282052</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KL_PVKL972 0</td>
</tr>
<tr>
<td>cs-uri-query</td>
<td>Query String</td>
<td>The query string portion, if any, of the URI requested by the client</td>
<td>PAGENAME=/about_us/index.swf&amp;v</td>
</tr>
</tbody>
</table>
P3P Considerations for Third-Party Page Tagging

Conceptual information about third-party tagging and preventing cookie-blocking using P3P.

• Understanding Third-Party Page Tagging
• Using P3P to Prevent Cookie-Blocking

Understanding Third-Party Page Tagging

In most implementations, the Adobe persistent cookie is viewed as a first-party cookie. First-party cookies are defined as those associated with the host domain.

Suppose a user visits http://www.example.com/. Assuming that a Sensor is installed and operational on the web server hosting the domain, an Adobe persistent cookie is set and viewed as a first-party cookie. You may, however, want to collect measurement data from a third-party site through the use of embedded objects, which are requested and loaded from your server that is running Sensor instead of from the third-party server hosting the page or advertising program. For example, http://www.example2.com/ serves a web page with an embedded object request served from http://www.example.com/. The request for the embedded object from http://www.example.com/ results in a cookie being set; however, in this context, the web browser or user-agent views the cookie as a third-party cookie.

In newer web browsers such as Microsoft’s IE6, privacy features filter cookies based on their compact policies sent in the HTTP response header from the web server. In the default IE6 settings, which most users never change, third-party cookies are blocked when they have nonexistent or unsatisfactory compact policies. Most sites that are experiencing cookie-blocking problems have third-party cookies on their site that do not have the appropriate compact policies being sent in the HTTP response header. Additionally, some cookie-blocking problems occur when a site is framed by another site. For example, an online store that is part of an online shopping portal may appear in a frame provided by the portal. From the perspective of the browser, the store content may appear to be third-party content when framed by the portal. However, if a visitor goes directly to the online store without going through the portal, the content will be first-party content. Thus, the online store finds their cookies are blocked only when visitors come in through the portal.

Web-based mail systems cause a similar problem. If a website visitor emails a web page to a friend who uses a web-based mail system, the email message appears as third-party content to the friend’s browser because it is framed by the email system. If there are any cookies associated with the page that was emailed, they are treated as third-party cookies by IE6.

Using P3P to Prevent Cookie-Blocking

P3P provides a standard way for websites to encode their privacy policies in a computer-readable XML format. P3P-enabled web browsers and other P3P user agents automatically fetch P3P privacy policies, parse them, and compare them with a user’s privacy preferences.

To prevent IE6 from blocking cookies on your site, you need to ensure the following:

1. All of the cookies that are being set in a third-party context have P3P compact policies associated with them.
2. The appropriate compact policy is sent using a custom HTTP response header.
3. Those compact policies are considered satisfactory by IE6.
4. If the third-party cookies are being set by another company, you may need to ask them to enable P3P and set P3P compact policies. Any host that sets a P3P compact policy also must have a corresponding full P3P policy.
Users can change their IE6 settings so that cookies are blocked under other conditions as well; however, placing satisfactory compact policies on third-party cookies prevents most IE6 cookie-blocking.

The following is an example of such a P3P header:

```
P3P: policyref="http://www.myserver.com/w3c/p3p.xml", CP="NOI DSP COR PSA PSD OUR IND COM NAV"
```

In this example, the file `p3p.xml` is used to reference an associated `policy.xml` file residing on your web server that denotes the kinds of information your website collects, dispute resolution methods that your organization adheres to, how the data collected is used, who owns the data, and other standard information related to Internet Privacy. The three character codes following the “CP” are the compact policy codes that emulate what is stated within your `policy.xml` file.

All compact policies and policy XML files should be tailored for the respective organization for which they are being deployed, accurately specifying their internal privacy policies with regard to website data collection. A multitude of P3P policy editors can be found online along with more in-depth information relative to implementing an appropriate privacy policy within your website.

For more information on how Internet Explorer 6 handles P3P Headers, please visit:

# Page Tag Variable Mapping Worksheet

Tables containing list of constant and custom variables.

- **Constant Variables**
- **Custom Variables**

## Constant Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Short Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>ct</td>
<td>constant tag</td>
<td>MODIFIED WITHIN JS FILE</td>
</tr>
<tr>
<td>cd</td>
<td>constant domain</td>
<td>MODIFIED WITHIN JS FILE</td>
</tr>
<tr>
<td>cu</td>
<td>constant uri</td>
<td>MODIFIED WITHIN JS FILE</td>
</tr>
<tr>
<td>ce</td>
<td>constant end</td>
<td>MODIFIED WITHIN JS FILE</td>
</tr>
<tr>
<td>c</td>
<td>constant</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>sw</td>
<td>screen width</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>sh</td>
<td>screen height</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>cd</td>
<td>color depth</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>d</td>
<td>dynamic</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>dt</td>
<td>document title</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>dr</td>
<td>document referrer</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>cb</td>
<td>cache busting</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>v</td>
<td>visual custom</td>
<td>DO NOT SET</td>
</tr>
<tr>
<td>v_jd</td>
<td>visual custom</td>
<td>DO NOT SET</td>
</tr>
</tbody>
</table>

## Custom Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Variable Short Description</th>
<th>Additional Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>v_pn</td>
<td>Page Name</td>
<td>v_pn=“Application_Form”</td>
</tr>
<tr>
<td>v_pc</td>
<td>Page Category</td>
<td>v_pc=“Application_Process”</td>
</tr>
<tr>
<td>v_pe</td>
<td>Page Error</td>
<td>v_pe=“Validation_Error”</td>
</tr>
<tr>
<td>v_c</td>
<td>Click-through Campaign</td>
<td>v_c=“Campaign 1”</td>
</tr>
<tr>
<td>v_ca</td>
<td>Click-through Campaign Asset</td>
<td>v_ca=“72890ab”</td>
</tr>
<tr>
<td>v_cr</td>
<td>Click-through Campaign Referrer</td>
<td>v_cr=<a href="http://money.cnn.com/markets/&amp;v_cp=CNNMy_Mkt%E2%80%9D">http://money.cnn.com/markets/&amp;v_cp=CNNMy_Mkt”</a></td>
</tr>
<tr>
<td>v_ic</td>
<td>Impression Campaign</td>
<td>v_c=“Campaign 1”</td>
</tr>
<tr>
<td>v_ica</td>
<td>Impression Campaign Asset</td>
<td>v_ca=“72890ab”</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Variable Short Description</td>
<td>Additional Information</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>v_icr</td>
<td>Impression Campaign Referrer</td>
<td>v_cr=<a href="http://money.cnn.com/markets/&amp;v_cp=CNNMy_Mkt">http://money.cnn.com/markets/&amp;v_cp=CNNMy_Mkt</a>&quot;</td>
</tr>
<tr>
<td>v_ln</td>
<td>Link Name</td>
<td>v_ln=&quot;About%20Us&quot;</td>
</tr>
<tr>
<td>v_eurl</td>
<td>Exit URL</td>
<td>V_eurl=&quot;<a href="http://www.offsite.com/">http://www.offsite.com/</a></td>
</tr>
<tr>
<td>v_1</td>
<td>custom variable 1</td>
<td></td>
</tr>
<tr>
<td>v_2</td>
<td>custom variable 2</td>
<td></td>
</tr>
<tr>
<td>v_3</td>
<td>custom variable 3</td>
<td></td>
</tr>
</tbody>
</table>