SAS® Intelligent Advertising for Publishers

Request Application Program Interface
Request Application Program Interface

SAS® Intelligent Advertising for Publishers (SASIA) is a flexible advertising delivery platform able to serve online display, video, and mobile advertising channels. This Application Programming Interface (API) contains directives, protocols, and methods for working with and requesting ads with SASIA.

Web Server Ad Call Directives

As a special-purpose web server, the SASIA ad server responds to HTTP requests from any client. It uses path information to determine which creative to serve. The method in this path contains the directive that tells the server how to return the creative, as in the following example:

```
http://crtl.aimatch.com/{custid}/hserver/site=A/size=728x90
```

There are multiple server directives you can use to develop ad calls that are appropriate for your server model. In this case, the directive `/hserver/` determines the HTTP content type that the server will return. In the case of hserver, it is text/html. For a jserver, it would be application/x-javascript.

Ad Tag Parameters

The following ad tag parameters allow you to change simple functionality without revising your creatives. Note that ad tag parameters are not case sensitive (for example, `/AREA=4` is the same as `/area=4`).

**custid**

A unique string used to identify each SASIA customer. The specific custid value to be used for your implementation of the SASIA ad tag is provided by the SAS Implementation Manager.

The following ad call parameters are key-value pairs that the engine uses to select ads:

**area**

Passes in the area value for targeting.

**custom tag name**

Can be included in your ad calls for logging or targeting if you have created custom tags and associated values. Pass in the exact tag name and value. For example, if you have designated a tag called "position" that has three values (top, middle and bottom), the ad tag might contain `/position=top/`.

**site**

Specifies an available site value for targeting. Ads can respond to a call that contains the specified site name if they have been targeted to that site or if there is no site value specified.
size
specifies an available ad size value for targeting.

viewid
notifies the system about which ad requests belong together.

random
prevents the caching of creatives, especially to third-party servers.

nocompanion
prevents selection of a companion flight during an ad call.

keyword
specifies one or more keywords used in targeting. Separate multiple keywords with a space or comma.

duration
specifies the maximum time duration, in seconds, to be filled with multiple ads in a dserver ad request.

latitude\longitude
specifies the latitude and longitude, in degrees, to be used in proximity targeting.

saspb, pbfcid
used in passbacks, See Ad Passbacks.

Considerations for Developing an Ad Call

Choosing the best ad call for your situation depends on the targeting schema that you have created, the nature of your web pages, and the creative you are serving to the pages. Carefully consider these factors before choosing an ad call:

Caching and Cache Busting

Web browsers often save a local copy of files that they have requested and displayed in order to display them more quickly the next time it is requested. The saved version is used instead of re-requesting the file from its server. Some Internet Service Providers (ISPs), network proxy servers, or other network components might use the same process. The process is known as caching and it is normally a positive enhancement to the web-browsing experience.

When an online advertisement is cached and then viewed, the ad server is not able to count the impression. To prevent this problem, each instance of a given ad call should contain a unique random number. The unique number causes the browser or proxy server to see it as a unique object and therefore make a fresh request for it from the ad server. The result is proper
impression counts and revenue allocation. This process is known as "cache-busting", and there are several ways to effectively implement it. It is very important that cache-busting techniques be used properly.

**Cache-Busting Techniques**

SASIA uses a specific ad call parameter designed to receive the random cache-busting number (/random=), although any unique variation in the ad-tag string can provide effective cache busting. The advantage of using the /random= parameter is that it can be referenced in several ways by objects that are downstream from the ad request.

The parameter can be used to provide cache busting within the source code of rich media or third-party advertisements without a separate, unique-number generation process. The random number can be generated by a server-side process and dynamically inserted into the ad call, or it can be generated using a client-side script that generates the entire ad call dynamically. Caching can lead to major discrepancies between SASIA counts and those of third-party ad servers or site publishers.

The parameter prevents the caching of creatives, especially to third-party servers. You can prevent caching to maximize impression counts (rather than using a cached version of the ad from the browser or a proxy server). Any value passed in with the random ad call parameter is also used to populate the %%RANDOM%% token within ads that contain it.

It is recommended that each ad call on a page have a unique random value. If two or more ad calls on the page have the same targeting values and random values, the first ad delivered for rendering could be cached in the other ad call positions. In addition, the random value should be a different value for each page or page refresh. This is typically done dynamically using Javascript. Example:

```
http://crt1.aimatch.com/{custid}/hserver/site=x/area=y/size=728x90/pos=top/random=345678/
```

**Companion Ads**

The /viewid= tag is required if you are going to use companion ads (also called roadblock ads). The viewid parameter notifies the system which ad requests belong together. Using the same viewid value in a set of ad calls enables the system to treat the set of ad calls as a single page. The number specified as the viewid must be the same for each ad call in a group, which identifies a set of ad calls as existing on one page of a web site. Also, the number must be unique to that service of the group (or page). This is typically accomplished dynamically, using Javascript.

Specify viewid if you want to use tier settings to suppress duplicate flights or advertisers appearing on the same page. You can also use viewid if you want to suppress delivery of creatives from different advertisers appearing on the same page (according to their defined categories). Example:

```
http://crt1.aimatch.com/{custid}/hserver/site=x/area=y/viewid=438943894343/size=728x90/pos=top/
```
Maximum Request Length

The maximum size of a request is 32767 bytes. Anything longer will be truncated. There are no additional limits on the path info, query string or header portions of the ad call.

Ad Call Methods

iserver

returns a 302 RELOCATE response and the image source URL. This call is designed to be served to the `<img>` tag. For example, you could insert the iserver URL into the `src` attribute of the `<img>` tag. Example:

```
< img
src="http://crtl.aimatch.com/{custid}/iserver/site=x/area=y/viewid=438943/random=948576789/size=728x90" width=728 height=90 border=0>
```

hserver

returns the text source of a creative with the tokens replaced as defined in SASIA. This call is designed to be served to the SRC attribute of the `<iframe>` tag. Example:

```
< iframe
src="http://crtl.aimatch.com/{custid}/hserver/site=x/area=y/viewid=438943/random=1827473656/size=728x90">
```

jserver

returns JavaScript that dynamically writes the advertisement into the web page that called it. This call is designed to be served to the SRC attribute of the `<script>` tag. Example:

```
< script
src="http://crtl.aimatch.com/{custid}/jserver/site=x/area=y/viewid=438943/random=102958674/size=728x90">
```

bserver

is a special-purpose ad call that returns a bundle of advertisements with a single call. The system parses out the ad call parameters for each ad in the bundle, and then returns JavaScript code that contains the HTML code necessary for all of the ads. This code presents the ads as a group of numbered JavaScript variable strings. Example:

```
< script
src="http://crtl.aimatch.com/{custid}/bserver/ball/site=x/area=y/viewid=43892143/random=09874756/b1/size=728x90/b2/size=728x90">
```
requests zero or more ads whose combined duration is less than or equal to the value specified in the DURATION tag. As with other directives involving the DURATION tag, only creatives having their duration field set are eligible for serving. To fulfill serving this directive, the Engine internally makes a series of ad selections. Each of the selections starts at the top of the ad queue. Tier and flight settings operate as usual on every selection. After each selection, it reduces the remaining duration by the duration value of the creative just selected, and makes another selection attempt. The series of selections ends when a default ad is selected. The default is not returned in the response.

The Engine uses the same tags passed in with the ad call on every selection. One additional tag (ADPOS) is synthesized on each selection and its value varies with the order of the selection. The first selection is made with ADPOS=01, the second with ADPOS=LAST, and the remaining ones with ADPOS=02, 03, 04, etc. In this way, ads can be targeted to specified selection positions. Since the Engine cannot know ahead of time how many ads will be selected, it forces the second selection to be tagged as LAST, which then allows for targeting this special position.

The system does not presently create the ADPOS tag itself or its values, so they are not available in the user interface. If you want to use ADPOS, you must manually create it in the user interface with the values of 01, 02, etc., and LAST, in order to match the values synthesized by the Engine.

A viewid must be specified if certain ad selection features are desired, such as competitive exclusion, eliminate duplicates, companion ads, etc., as usual. That is, no viewid is synthesized automatically for the request bundle.

The response consists of a simple concatenation of all creatives. The second ad selected (tagged as LAST) is forced to be at the end of the response. The mime type provided is that of the first creative. If no ads are selected, the response body is empty. No default is returned.

**Beacons**

Beacons assist in understanding and monitoring customer behaviors and activity. The use of a beacon allows the site to record the actions of the user opening the page that contains the beacon and allows the event to be counted. Beacons are a convenient way of gathering statistics. SASIA supports several types of beacons that can be used in different contexts.

**Count Directives**

SASIA has a generic event counting service that can deliver a response from a request based on the creative selected to be served. In SASIA, actions are sent to the system using the count directive. For example:

```
http://{SASIA adserver URL}/{customer short name}/{content type}/count/fcid=1234/act=3/inc=1
```

increments the action count for fcid 1234 [the unique creative identifier] by 1.
act=1

denotes a beacon count URL.

Example:

http://crtl.aimatch.com/{custid}/count/act=1/fcid=57/site=x/area=y

act=2

denotes a click count URL. When using this instead of the 'adclick' directive the click count is incremented but the browser is not relocated to the click URL destination.

act=3

denotes a generic action count URL. For more information on standard and custom action tracking, refer to Action Tracking.

act=4

denotes a viewed URL.

act=5

supports optimization methods. This feature is not enabled by default.

inc=X

increments the count by X.

When act= is missing, 1 is assumed. When inc= is missing, 1 is assumed.

Other targeting tags present in the URL (for example: site or area) are logged against when the URL is executed. Note that the current version of action tracking is not cookie-based, hence the need for the FCID value to be present in the URL for correct counting.

The following example increments the click count for flight creative id 57 (fcid=57) and relocates to the click destination specified for that creative:

http://crtl.aimatch.com/{custid}/adclick/fcid=57/site=x/area=y

This example increments the click count for flight creative id 57 but does not relocate to the click destination:

http://crtl.aimatch.com/{custid}/count/act=2/fcid=57/site=x/area=y
Action Tracking

Standard action tracking (for example, FCID, ADVID, ACTID) allows you to associate actions with a creative for the advertiser who owns the creative. The creative can be recently seen or clicked by a web site visitor. This feature is useful for monetizing behaviors such as leading a web site visitor from an advertisement to a landing page where a transaction (such as a product purchase) subsequently occurs. The action-tracking URL needs to be placed on the action success page (the landing or results page) so that when the page is reached by the visitor, the URL can notify SASIA to count it. Standard action tracking requires a value be entered in the Cost Per Action field in the Goals and Revenue section of the Edit Flight page.

The following example displays the syntax for an action-tracking URL associated with a viewed or clicked creative:

```
http://{adserver domain}/{custid}/count/act=3/fcid={FCID}
```

where {FCID} is replaced with the creative's FCID value. The code /act=3 tells SASIA that this is a standard action-tracking, counting URL. The FCID value can be found in the user interface next to the name of the creative at the bottom of the Edit Flight page.

The following example displays the syntax for an action-tracking URL associated with an advertiser whose creative has been seen or clicked:

```
http://{adserver domain}/{custid}/count/act=3/advid={ADVID}
```

where {ADVID} is replaced with the advertiser's ADVID value. The ADVID value can be found in the URL of the Edit Advertiser page.

Site and area values are not needed in the action-tracking URL because they are captured at the time of the impression or click that preceded the action.

For the default action (act=3), the ad server automatically tracks impressions and clicks for any flight that has a CPA set to a value > 0.00. Flights without a CPA or with CPA=0.0 are not tracked.

The following action-tracking URL:

```
http://{adserver domain}/{custid}/count/act=3/advid=50
```

examines the visitor's history of tracked flights for advertiser 50. The flight creative with the most recent click or impression is credited with the action. If a flight has multiple creatives with recent clicks or impressions, the one with the most recent click is credited. Clicks that are older than 15 days are ignored. Impressions older than one day are ignored. If there are no flights with a qualifying click or impression, then the action is ignored.

Custom action tracking enables you to create and track custom actions (for example, tracking whether a web site visitor has viewed 50% of a video). Custom action tracking requires
Impressions are tracked for each flight with an action policy that covers impressions or clicks. Flights without an action policy are not tracked unless the action-tracking URL contains a valid FCID value, in which case the custom action is logged even without a Custom Action Policy enabled for the flight. The active period for impressions or clicks are controlled by the action policy that is set for each action on the flight. Each action has its own policy that governs impressions and clicks per flight.

An action is triggered by a COUNT directive that references an advertiser and a custom action. The custom action is referenced by ID or name. The flight creative that has the most recent active tracked impression or click is credited. If the flight has both a click and an impression, the more recent of the two determines which is credited unless the policy has the Click trumps impressions option set, in which case the click is credited regardless of which is more recent.

Counts for custom actions are logged by date, FCID, action ID, attribution type, site, and area. Attribution type is either 0 (for impression) or 1 (for click). Site and area are logged as the original site and area that were associated with the credited impression or click when it occurred. Custom Actions must be associated with the advertiser in order for the Custom Action to appear in the Action Policies dropdown menu in the Edit Flight screen.

The custom action URL can use either the custom action's actid value or its name (actname). If both actid and actname are present, then the actname is ignored. To find the actid value for a custom action, edit the custom action and check the URL for the Edit Action page.

```
http://crtl.aimatch.com/customer/count/advid=12/actid=4
```

This URL searches the visitor's history for a flight associated with advertiser 12 that has the most recent click or impression. Only flights that have policies for the action with the actid=4 tag are considered. The act=3 tag is not required.

```
http://crtl.aimatch.com/customer/count/advid=13/actname=videostart
```

This URL searches the visitor's history for a flight associated with advertiser 13 that has the most recent click or impression. Only flights that have policies for the action with the name "videostart" are considered.

There are some pre-defined actions available in the user interface and you can create additional custom actions in the Custom Actions tab under the Traffic tab.

**Ad Passbacks**

During normal ad selection, the engine remembers the path information, the querystring, and blacklisted advertisers of the last 10 ads that were served in the last five seconds, per visitor (configurable by customer). All ads are treated as potential passbacks; no special trafficking or configuration is required. If the same FCID is served multiple times in a short time span, only the last ad served is kept.
Passback behavior is triggered by a network redirecting to SASIA with a tag that includes SASPB and the FCID of the original ad served (for example, ".../hserver/SASPB/FCID=1234"). If the FCID is absent, invalid, or exists for an ad that is not in the visitor's recent queue, then a default ad is served and no true passback is possible. The passback tag and corresponding FCID value can be found by selecting the green button beside the creative that is listed on the Edit Flight page.

The SASPB/FCID passback tags that are sent by the network are ideally supplied by a format template that creates them in response to the original ad served (if the network is "cooperative"). Otherwise, they must be hardcoded in the trafficking of SASIA passbacks in the network itself, usually by size.

Assuming the passback FCID is for a valid ad recently served to the visitor, its associated path information, query string, and blacklisted advertisers are resurrected for the passback ad selection. The ad's advertiser is added to the blacklisted advertisers for this ad selection only. Ad selection starts at the top of the queue and blocks selecting any flight associated with the blacklisted advertisers. Appropriate trace messages are shown when encountering such flights.

If the same FCID is passed back multiple times in a short time span, each passback is likely to resurrect the same set of tags for the passback ad because only the most recent pathinfo and other information is retained. This is expected to occur infrequently, and only when a page has multiple ads of the same size. In such cases, the ad tags are not likely to be very different. If this becomes an issue, prudent use of frequency capping at the creative level can mitigate its effect.

When serving an ad in response to a passback, the impression count for the original ad selection is decremented (uncounted), using its timestamp, tags and other information, unless it was not in the visitor's queue. Frequency capping is ignored. No attempt is made to credit a capped creative-flight pair when this creative is uncounted.

The ad served in response to a passback is counted normally, using its timestamp and the original ad's tags. The passback itself is not counted. That is, there is currently no separate counting of passbacks that easily enables the reporting of fill rate.

The ad that is served in response to a passback is sampled normally and includes a column that references the passed back FCID. During simulation, a sampled impression that was served in response to a passback is ignored because it is a duplicate of the sample record of the original impression. During unique visitor (UV) analysis of sample data, no attempt is made to reconcile passbacks with their original flights. So UV results for a flight includes all visitors who were served that flight, regardless whether all serves of that flight were passed back.

**Supertags**

A SUPERTAG is a special tag that can be included in an ad call and expanded into one of any number of arbitrary collections of other tags and values. Supertags can be used to ease a customer's migration from other adservers, and provide a level of tag management to ad operations. Supertags provide a way of mapping a value to an arbitrary pathinfo snippet. For example, a supertag value "HOMETOP" could be defined as "AREA=HOME/POSITION=TOP/SIZE=LEADERBOARD". Then the following ad call:

http://crtl.aimatch.com/customer/hserver/supertag=hometop
Any number of supertag values can be created to pre-define multiple sets of tags and values. When an ad call includes "SUPERTAG=supertagvalue", the tags in the corresponding pathinfo are added to the ad request internally. The "SUPERTAG" tag name itself is not configurable.

Each hserver or jserver ad call can reference SUPERTAG only once. If multiple references to SUPERTAG are present, the last one wins, as with other tags. For bserver, SUPERTAG can appear once in each numbered B section, assuming it didn't also appear in the BALL section. Like other tags, if SUPERTAG appears in the BALL section, it becomes a tag for all ads.

Multiple values can be supplied for SUPERTAG, separated by commas, similar to other tags. The resulting substitution yields the union of all the values' definitions, with later ones taking precedence. For example, a supertag value "LOCALNEWS" could be defined as "SITE=NEWS/AREA=LOCAL". In combination with the "HOMETOP" definition from the above example, the following ad call:

http://crtl.aimatch.com/customer/hserver/supertag=hometop,localnews

would be equivalent to:

http://crtl.aimatch.com/customer/hserver/position=top/size=leaderboard/site=news/area=local

Note that since both supertag values included a substitution for AREA, the last one's value was used. An ad call that reverses them:

http://crtl.aimatch.com/customer/hserver/supertag=localnews,hometop

would be equivalent to:

http://crtl.aimatch.com/customer/hserver/site=news/position=top/size=leaderboard/area=home

An ad call that includes SUPERTAG can freely include other tags as well, and the complete set of tags implied by that mix determines what ad is served. When a SUPERTAG value maps to a pathinfo that includes a tag that is also explicitly included in the ad call, the explicit value from the ad call takes precedence. For example, given the HOMETOP example above, the following ad call:

http://crtl.aimatch.com/customer/hserver/size=728x90/supertag=hometop

results in requesting an ad with size 728x90--not LEADERBOARD. This is true regardless whether the supertag is expressed before or after the size tag in the ad call.

For this and other reasons, the definition of a supertag value as a segment of pathinfo cannot be taken too literally. The supertag value is not literally dropped into the ad call URL and processed as if it came in that way. The pathinfo definition cannot, for example, be used to alter the number of ads returned by a bserver call.
The pathinfo associated with a supertag value can include any type of tag used for ad targeting, including site, area, size, custom tags, duration, keyword, and even NOCOMPANION. In theory, it could also include VIEWID, PID, TRACE, and FCID, although the utility of such tags in supertags is suspect.

Supertag pathinfo can also include one nested SUPERTAG. In other words, one supertag value can reference another. Nesting can be to any level, although the mental complexity of managing such a configuration should realistically limit the user to 2 or 3 levels. The tags produced by higher levels take precedence over tags at the deeper levels, just as in the example above where the explicit tags in the ad call take precedence. In click responses, the CLICKURL returned includes the original SUPERTAG reference. This is also true for the Beacon Count sent for those ads.

Like other tag values, supertag values are case-insensitive. Unlike other tag values, there are no restrictions on use of punctuation, etc. Special characters only need to be URL-encoded appropriately. This relaxation is to maximize compatibility when supertags are used to help with migration from other ad servers.

There is no logging of supertag values. If you need a log, create another conventional custom tag and reference it from within each supertag value's pathinfo. Supertags are not included in sample files, so redefinition of supertag values has no effect on simulations.

Namespacing

When a SUPERTAG value has an embedded dot (period, "."), the string in front of the dot defines a namespace for the value. Different namespaces can define the same value. For example, the following values can all be defined separately:

- ABC defined as "/AREA=XYZ"
- GENRE.MA defined as "/KIDS=NO/THEME=ADULT"
- PROGID.ABC defined as "/NAME=THISISAPROGRAM/GENRE=MA"

Supertag lookups are done in either of two ways:

- The ad call includes "SUPERTAG=value".
  - If the value includes a namespace, the lookup is done through that namespace.
  - When the value doesn't include a namespace, the functionality is equivalent to the current Supertag functionality.
- The ad call includes "namespace=value".
  - The lookup is done through the namespace using the value. This is equivalent to SUPERTAG=namespace.value.

For example, both of the following result in the same lookup being done:

- SUPERTAG=GENRE.MA
- GENRE=MA

Supertag lookups can be recursive. For example, using the above definitions, an ad call with "PROGID=ABC" results in an expansion of "/NAME=THISISAPROGRAM/GENRE=MA/KIDS=NO/THEME=ADULT", because it first expands to "/NAME=THISISAPROGRAM/GENRE=MA", and then GENRE gets expanded.
A namespace is completely arbitrary and requires no configuration. It does not need to be a first class targetable tag, but it could be, in which case it serves both its targeting function and a Supertag expansion function.

**Supertag Use in Migrations**

Supertags can play an important role in easing migration from another ad server. Some require a publisher to tag pages referencing the name of an ad "slot" or "position". On their backends, those ad servers know that a specific slot value maps to a certain ad size, page position, and in some cases, other supplemental targeting.

You can minimize the work involved by defining a supertag value in SASIA for each slot known by the previous system. For each, the supertag value is set to the same name as the slot, and the pathinfo is set to the equivalent set of tags and values in SASIA. Most commonly, a slot maps to a specific site, area, and size. Then, when retagging pages for SASIA, you can minimally exchange references to your former server's slot names for SASIA ad calls referencing SUPERTAG, or, in some cases, use SAS-supplied Javascript to reference those names.

**Supertag Use for Tag Management**

Supertags also give you a means to change information in your ad calls without actually changing code on your web pages. For example, you can assign each unique ad/page combination a supertag value, and the ad call made for each would simply request SUPERTAG=supertagvalue. You can control the appropriate size and targeting by defining the pathinfo needed for each supertag value, and then change these freely over time.

**Content Personalization**

SASIA provides a system for creating, managing, and personalizing digital content and settings.

**SETTAG**

The SETTAG directive is used to set one or more cookies holding supplemental targeting tags. Once set, these cookies provide additional key/value pairs with every subsequent ad request, click, or action. Each cookie can be set to expire after an arbitrary time. You can use SETTAG:

- As an alternative to user reg. You can call SETTAG after a visitor logs on and provide whatever tags/values are associated with that visitor. The tags and values persist with the visitor's browser and are sent on every subsequent request. There is no need to store visitor data on the back end.
- To achieve cookie targeting. You can set tags via SETTAG when a visitor hits certain spots on their site, and then target ads against those values.
- To set a visitor's latitude and longitude, eliminating the need to supply them in every call.
- To test a live page engaging with certain tags that cannot otherwise be provided. For example, calling SETTAG with GEO_COUNTRY=AUS causes subsequent ad calls to override the GEO_COUNTRY tag that was looked up from the visitor's IP address.
• To prevent counting impressions, clicks, and actions resulting from testing by SAS or a publisher. This is done by providing the NOLOG tag in the SETTAG call.

The syntax for SETTAG is:

http://whatever.aimatch.com/customer/SETTAG/NAME=cookieName/TTL=secondsToLive/TAGS=urlEncodedTags

where:
• **cookieName** is the name for a set of tags. A cookie with this name, prefixed by "VT_" is set on the visitor. If absent, this directive is ignored.
• **secondsToLive** is the time-to-live for this cookie, in seconds. For example, TTL=86400 sets the cookie to expire in one day. If absent, the cookie is set to expire at the same time as the visitor's guid/mid (one year by default).
• **urlEncodedTags** is the set of tags and values to persist in this cookie. These must be URL-encoded and in pathinfo format (i.e., using encoded forward slashes to separate tags). If absent, this cookie is deleted. Use %3D for "=" and %2F for "/".

Some examples:

• http://whatever.aimatch.com/customer/SETTAG/NAME=smith/TTL=3600/TAGS=geo_country%3Dfra
  sets a cookie named VT_SMITH that expresses GEO_COUNTRY=FRA. All ad requests, etc., for the next hour overrides the visitor's natural GEO_COUNTRY lookup and instead provide FRA as that tag's value.
• http://whatever.aimatch.com/customer/settag/name=location/ttl=900/tags=latitude%3D89.9%2Flongitude%3D-43.21
  sets a cookie named VT_LOCATION that supplies the visitor's latitude/longitude in all requests for the next 15 minutes.
• http://whatever.aimatch.com/customer/SETTAG/NAME=nolog/TAGS=nolog
  sets a cookie named VT_NOLOG that blocks counting all ad requests, etc., for the next year.
• http://whatever.aimatch.com/customer/settag/name=somenetwork/TAGS=nettag1%3D342%2Fnettag3%3D2321%2Fnettag3%3D2935
  sets a cookie named VT_SOMENETWORK that expresses three tags, NETTAG1, NETTAG2, and NETTAG3. In this example, these might be tags that aren't needed by the Engine for targeting, but could be used as generic tokens (e.g., %%NETTAG1%%) in a template making an ad call to a network.
• http://whatever.aimatch.com/customer/SETTAG/NAME=smith
  clears the VT_SMITH cookie.

Multiple cookies can be set for a visitor by calling SETTAG multiple times, each time with a different cookie name. The combined contents of all such cookies are sent to the Engine as supplemental tags in the request. These tags override any tags held with the visitor within the Engine (for example, those produced from geo lookups, device lookups, and user reg). But tags that are explicitly sent in with an ad request override tags provided in these cookies. If more
than one cookie expresses the same tag, it is undefined which one takes precedence.

A tag=value pair included in an ad request URL’s pathinfo string which matches the SETTAG name but has a different value overrides the SETTAG value.

**SETTAG Persist**

If the SETTAG request uses the "/name={cookieName}" directive, the visitor's values are stored with the cookie, otherwise the values are stored in the visitor's session and discarded after the session.

If not using "/name={cookieName}", adding "/persist=1" to the SETTAG request saves the values beyond the visitor’s session. The values' lifespan is determined by the "/ttl={X seconds}" directive in the SETTAG request.

**Multiple Values for a Single SETTAG**

SETTAG supports multiple tag values for a single tag in the SETTAG. For example:

http://whatever.aimatch.com/customer/settag/name=multi/TAGS=sports%3Dhockey,football

creates a cookie 'VT_MULTI=hockey,football'. The visitor would be assigned the tag/values of 'sports=hockey,football'.

**SETTAG Append**

Values for a tag can be appended in a single cookie using the following sequence:

- /settag/tags=geo_country=usa
- /settag/append=1/tags=geo_country=gbr

The visitor would first be assigned the tag/value of geo_country=usa, and then the following request with APPEND=1 specified would assign the visitor the tag/values of geo_country=usa,gbr. The APPEND option can be used in conjunction with the TTL option detailed above.

**Stacked Tags**

SASIA supports tag values that allow for "stacked" values. Such a tag has multiple values, taken one at a time in subsequent ad requests. Once the tag is set this way, the first ad request evaluated uses the first value (or values, if multivalued). The next ad request for the same visitor uses the second value, and so on. After using the last value, it cycles around and uses the stack again, starting again with the first value.

Any tag can be given stacked values by prefixing the values with "Unable to render embedded object: File (" (exclamation mark) and separating them with more ") not found." For example:

/offer=!beer!soda!nuts

results in:

/offer=beer in the 1st request
/offer=soda in the 2nd
Multiple values are supported for each place in the stack, as appropriate for whatever tag is involved, using commas to separate the multiple values, as normal. For example: 
\[ /\text{offer}=!\text{beer},\text{soda},!\text{nuts} \]
results in: 
\[ /\text{offer}=\text{beer},\text{soda} \text{ in the 1st request} \]
\[ /\text{offer}=!\text{nuts} \text{ in the 2nd} \]
\[ /\text{offer}=<\text{empty}> \text{ beyond that.} \]

An empty position in the stack can be expressed with multiple "!". For example: 
\[ /\text{offer}=!\text{beer}||\text{soda} \]
results in: 
\[ /\text{offer}=\text{beer} \text{ in the 1st request} \]
\[ /\text{offer}=<\text{empty}> \text{ in the 2nd} \]
\[ /\text{offer}=\text{soda} \text{ in the 3rd} \]
\[ /\text{offer}=<\text{empty}> \text{ beyond that.} \]

Multiple tags can be given stacked values and they advance through their stacks together. For example: 
\[ /\text{offer}=!\text{beer}!\text{soda}!\text{nuts} /\text{pos}=!1!2!3 \]
results in: 
\[ /\text{offer}=\text{beer}/\text{pos}=1 \text{ in the 1st request} \]
\[ /\text{offer}=\text{soda}/\text{pos}=2 \text{ in the 2nd} \]
\[ /\text{offer}=\text{nuts}/\text{pos}=3 \text{ in the 3rd} \]
\[ /\text{offer}=<\text{empty}>/\text{pos}<\text{empty}> \text{ beyond that.} \]

The stacked values persist with the visitor for the balance of the session. If the session expires before all values have been used, a subsequent session does not resume where the last one left off. If the start of a new visitor session causes the tag to again be set to stacked values, that stack starts being used from its beginning.

Any existing method that is supported for setting tags is eligible to set a tag to stacked values, including SETTAG, user reg, SUPERTAG, or the ad request itself.

When a tag has already been set to stacked values, an ad request can override it with an explicit value, much as an explicit value can temporarily override a geo lookup, for example. The override doesn't affect the stack. If the next ad request does not include the tag, the next stacked value is used.

When a tag has already been set to stacked values, if the tag is set again to another stack, then the old stack is discarded and the new one starts being used, even when two stacks express the same values.

For this reason, if SETTAG sets a tag with stacked values in a cookie, then every incoming request resets the stack because of the cookie's inclusion of that stack definition in every request. The result is that the stack becomes ineffective for anything but bundled ad requests, because it is constantly being reset. For bundled requests, the first ad uses the first value, the second uses the second, and so on.

If SETTAG is used to set tags in the session (and not a cookie), any stacked tag is set only once. If
SETTAG is used to set tags with Cassandra persistence (and not a cookie), any stacked tag is set once when the SETTAG occurs, and then again each time that visitor starts a new session.

An ad's beacon (vericount) URL and click URL contain the stack value for each stacked tag that was part of that request. If the tag having the stack value is referenced from a token, the stack value is used for the replacement.

When a passback occurs, if the original ad had been served using a stack value, the new ad served uses the same value for that tag and does not disturb the stack.

**Building an Ad Call Using Client-Side Javascript**

Because modern browsers ordinarily parse Javascript, you can create dynamic ad calls on the page for each of the methods described in "Ad Call Methods." All examples here are shown using client-side Javascript for building the ad call dynamically. Comparable ad calls can be created using server-side dynamic solutions such as ASP, ColdFusion, PHP, and so on.

**iserver**

The iserver ad call method is used to return a 302 RELOCATE response and a relocation URL to an "<img src=>" element on the page. This ad call is often used in email newsletters as most email readers do not accept the hserver | jserver | bserver methods.

**Strengths**

- Simplicity
- Enables ad delivery into email newsletters

**Weaknesses**

- Does not support rich media creatives; supports static image creatives only.
- Does not allow the insertion of a unique identification number for the creative ("FCID"). The FCID positively correlates clicks into the "<a href=>" URL at any time after the creative has been served. Omitting an FCID can result in incorrect click-throughs. Email newsletter content management systems typically allow the use of variable tokens such that a unique mid value can be placed in the <img src=> and <a href=> URLs.

**Example**

**Notes:**

- Replace {path to your adserver} with the correct adserver domain URL, including your specific {clientid} value.
- Replace {ad call targeting} with the correct targeting parameters for each ad instance.
- Replace the width= and height= values with the correct sizes.
Email Example

Notes:

- To avoid encoding issues with some email readers, insert the targeting values in the
  querystring using an ampersand ("&") as the delimiter between each key=value pair.

- The publisher’s content management system that generates the email needs to
generate two sets of values in order for SASIA to accurately pair the ad image with the
click-through. These values are mid={value} and pid={value}.

- For multiple ad calls in each newsletter, the mid value should remain the same for all ad
calls, and the pid value should be unique for each set of `<a href>` and `<img src>` tags.

- Add `hint=x` to the targeting string to avoid possible issues with intra-session routing to
  the ad server.

hserver

The hserver ad call method is used to return content to an `<iframe src>` element on the page.
The iframe is defined in the parent document, but is told to reference SASIA (with the
appropriate ad call parameters) for its content. The system returns the advertisement as a
complete document.

Strengths

- Enables the insertion of rich media (file-based advertisements).

- Not likely to delay the rendering of the page content because modern browsers usually
  render iframes asynchronously.

- Enables the insertion of a unique identification number (FCID) that positively correlates
  clicks at any time after the creative has been served. There is no expiration on click
  functionality.

- Supports all creatives that can function in the browser.
Weaknesses

- Does not support creatives that resize themselves or modify their position on the page without additional coding on the page.
- Can cause issues with creative that are designed to open in the parent page and not as a new document.

Example

Notes:

- Replace `{path to your adserver}` with the correct adserver domain URL, including your specific `{clientid}` value.
- Replace `{ad call targeting}` with the correct targeting parameters for each ad instance.
- Replace the `iframe width= ` and `height= ` values with the correct sizes.
- If you are using Minify Javascript remove all HTML comments beginning with `<!--`.

```html
<script type="text/javascript" language="JavaScript">
<!-- Hide from old browsers
// Modify to reflect site specifics
adserver = "http://{path to your adserver}";
target = "/{ad call targeting}";

// Cache-busting and viewid values
random = Math.round(Math.random() * 100000000);
if (!pageNum) var pageNum = Math.round(Math.random() * 100000000);

document.write('<iframe src="' + adserver + '/hserver/random=' + random + ' + target + '
/viewid=' + pageNum + '">

</iframe>

// End Hide -->
</script>
```

jserver

The jserver ad call method is used to return content to a `<script src>` element on the page. The `<script src>` ad call with is made and the system returns the selected ad wrapped in JavaScript, which is then written dynamically into the page and parsed.

Strengths

- Enables the insertion of rich media (file-based advertisements).
• Enables the insertion of a unique identification number (FCID) that positively correlates clicks at any time after the creative has been served. There is no expiration on click functionality.

• Supports all creatives that can function in the browser.

Weaknesses

• Javascript could delay the rendering of the rest of the content if the response of the ad server is degraded. See Building an Ad Call Using Krux PostScribe for more information.

• Poorly written or malformed advertisements might interfere with or (in extreme cases) break page content.

Example

Notes:

• Replace \{path to your adserver\} with the correct adserver domain URL, including your specific \{clientid\} value.

• Replace \{ad call targeting\} with the correct targeting parameters for each ad instance.

• Replace the iframe width= and height= values with the correct sizes.

• If you are using Minify Javascript remove all HTML comments beginning with <!--.

```<script type="text/javascript" language="JavaScript">
<!-- Hide from old browsers
// Modify to reflect site specifics
adserver = \"http://\{path to your adserver\}\";
target = \"/{ad call targeting}\"; 

// Cache-busting and viewid values
random = Math.round(Math.random() * 100000000);
if (!pageNum) var pageNum = Math.round(Math.random() * 100000000);

document.write('<script src="' + adserver + '/jserver/random=' + random + target + '/viewid=' + pageNum + '">
' + document.write('</script>');

// End Hide -->
</script>
```
**bserver**

The bserver ad call is the most efficient ad call. It has the smallest code footprint for multiple ad calls on a page and requires only one request to the system to get all ad content for the page. The single request aspect of the ad call also eliminates a "race condition" that can occur when multiple ad calls are made from the page simultaneously. A race condition can impair companion ad serving or sequential ad delivery.

The ad call construction shown below is required on the page only once, although you can use multiple bserver calls if you want to use multiple viewid values on a single page. The ad-rendering code for each ad (also shown below) is inserted at each ad position on the page.

**Strengths**

- Most efficient method of delivering multiple ads to a page.
- Enables the insertion of rich media (file-based advertisements).
- Enables the insertion of a unique identification number (FCID) The FCID positively correlates clicks at any time after the creative has been served, so there is no expiration on click functionality.
- Prevents a race condition from occurring.

**Weaknesses**

- Javascript could delay the rendering of the rest of the content if the response of the ad server is degraded. See Building an Ad Call Using Krux PostScribe for more information.
- Poorly written or malformed advertisement code might interfere with or (in extreme cases) break page content.

**Example**

**Notes:**

- Replace `{path to your adserver}` with the correct adserver domain URL, including your specific `{clientid}` value.
- Replace `{ad call targeting}` with the correct targeting parameters for each ad instance.
- If you are using Minify Javascript, remove all HTML comments that begin with `<!—`. 
- Add the adx Javascript variable and its accompanying `/bx/` string for each ad instance on the page, replacing the `x` with an incremented number.
- For each adx variable created, add an `adx` + string in the section of Javascript, which builds the `script src` bserver ad call (shown below).
- Edit the rendering code to replace `x` with the correct `/bx/` value that was referenced in
the ad call.

**Note:** The initial value of \( x \) cannot be zero (0).

- The resulting bserver URL pathinfo (the ad call request itself, not the Javascript used to create it) cannot exceed 2048 characters.

- `/ball/` strings are appended to each `/bx/` ad request section of the bserver URL, which gives you the ability to insert global ad call data into each ad request while defining it only once.

- Typically, this would be used for random and viewed values, but it can also include targeting tags (for example, if all ad calls on the page use the same SITE and AREA values). The `/ball/` string should appear after `bserver` in the ad call, but before any `/bx/` strings. If the ad call is not set up correctly, the system will not fail, but it might include extraneous strings in the ad requests.

- If the number of `document.writeln(bx)` calls on the page do not match the number of `/bx/` ads defined in the bserver script src call, the impressions are logged (even though the ads are not rendered). Make sure that the `bx()` calls match the `/bx/` ads defined in the bserver script src call.

- You can add multiple bserver calls on a page but do not duplicate `/bx/` values between the calls.

- Because the `/ball/` string contains a `/random=` value that is appended to each `/bx/` ad request, multiple ad requests to the same third party ad server within the same bserver ad call will contain identical `/random=` cache-busting values. The identical values might affect the third party ad server's impression counts. One alternative is to use multiple bserver ad calls on the page.

- The ad call below uses the `aimRnd` variable value for both the `/random=` and `/viewid=` values.

- Rich media file-based ads should be stripped of all comment strings.

- There are known issues with Google ads when multiple Google ads are served to the same bserver call. Test the multiple Google ads with the bserver ad call in a test campaign and test environment before going live.

```html
<head><script language="javascript" type="text/javascript"> <!--
var b1 = "";
var b2 = "";
var b3 = "";
var b4 = "";
//-->
</script>
<head>
<body>
```
At the Ad Locations -- insert the following Javascript (edit “bx” to reference the appropriate ad by number):

```javascript
<script type="text/javascript" language="JavaScript">
document.writeln(bx);
</script>
```

**Building an Ad Call for Email Delivery**

The following is an example of using an iserver ad call method to build an ad call for email delivery.

**iserver**

The iserver ad call method is used to return a 302 RELOCATE response and a relocation URL to an "<img src="" element on the page. This ad call is often used in email newsletters as most email readers do not accept the hserver | jserv | bserver methods.

**Strengths**

- Simplicity
- Enables ad delivery into email newsletters
Weaknesses

- Does not support rich media creatives; supports static image creatives only.
- Does not allow the insertion of a unique identification number for the creative ("FCID"). The FCID positively correlates clicks into the "<a href=>" URL at any time after the creative has been served. Omitting an FCID can result in incorrect click-throughs. Email newsletter content management systems typically allow the use of variable tokens such that a unique mid value can be placed in the `<img src=>` and `<a href=>` URLs.

Example

Notes:

- Replace {path to your adserver} with the correct adserver domain URL, including your specific {clientid} value.
- Replace {ad call targeting} with the correct targeting parameters for each ad instance.
- Replace the width= and height= values with the correct sizes.

< a href="http://{path to your adserver}/{custid}/adclick/{ad call targeting}">< img src="http://{path to your adserver}/{custid}/iserver/{ad call targeting}" width={width} height={height} border=0>

Email Example

Notes:

- To avoid encoding issues with some email readers, insert the targeting values in the querystring using an ampersand ("&") as the delimiter between each key=value pair.
- The publisher’s content management system that generates the email needs to generate two sets of values in order for SASIA to accurately pair the ad image with the click-through. These values are mid={value} and pid={value}.
- For multiple ad calls in each newsletter, the mid value should remain the same for all ad calls, and the pid value should be unique for each set of `<a href>` and `<img src>` tags.
- Add hint=x to the targeting string to avoid possible issues with intra-session routing to the ad server.

< a href="http://{path to your adserver}/{custid}/adclick?hint=x&site=x&area=y&size=728x90&mid=1234&pid=5678">< img src="http://{path to your adserver}/{custid}/iserver?hint=x&site=x&area=y&size=728x90&mid=1234&pid=5678" width={width} height={height} border=0>< /a>
Building an Ad Call Using Krux PostScribe

Remote scripts, especially ads, can block a page from doing anything else while they load. They can contribute to load times, which affects your bottom line. *Asynchronous* ads do not block the page and can be delivered after core content. However, asynchronous ads may contain calls to `document.write`, which expects to be handled synchronously.

PostScribe allows you to deliver a synchronous ad asynchronously without modifying the ad code. PostScribe uses DOM Proxies, a way to ensure that content is written as close to the way the browser would natively write content with `document.write/innerHTML`. Therefore, it behaves as the browser would, without complex parsing or hacks.

**iframe Example**

The following example uses iframe and merely demonstrates a simple way to use the PostScribe functionality. Feel free to use other patterns to create your ad call.

**Notes:**

- `aimRnd` is defined as a random number to prevent browser caching.
- `adserver` is defined as the base URL of all the request on the page. Replace `adserver` to match your account specifics.
- Replace `{your domain}` with the correct adserver domain URL, including your specific `{clientid}` value.
- The variable `allAdTags` is defined as a variable to hold the common parameters that all the ad requests on this page will use.
- In this example, the full ad call DOM object is built as an iframe and stored as a string in a variable. Previously declared variables are used while building the string.
- At each place in the content where you want an ad, use a `div` tag as the container object. Inside the tag is where the PostScribe library is called, the `div id` is passed, and the ad call identified. Ensure the `div id` matches the first parameter in the PostScript call.
- You can download `htmlParser.js` and `postscribe.js` from the krux/postscribe GitHub site.

```html
<html>
<head>
<script src="http://{your domain}/path/to/htmlParser.js"></script>
<script src="http://{your domain}/path/to/postscribe.js"></script>

<script>
var aimRnd = Math.round(Math.random() * 100000000);

// adserver URL
adserver = "http://{your domain}/path/to/hserver";
```
// Ad tag targeting values which will be appended to each ad request section in the bserver ad call

allAdTags = "site=testpages-area=async/random" + aimRnd + "/viewid=" + aimRnd;

b1 = '<iframe src=
"' + adserver + '/' + allAdTags + '/size=728x90?" width="728" height="90" scrolling="no"></iframe>'';

b2 = '<iframe src=
"' + adserver + '/' + allAdTags + '/size=300x250?" width="300" height="250" scrolling="no"></iframe>'';

b3 = '<iframe src="' + adserver + '/' + allAdTags + '/size=160x600?" width="160" height="600" scrolling="no"></iframe>'';

b4 = '<iframe src="' + adserver + '/' + allAdTags + '/size=728x90?" width="728" height="90" scrolling="no"></iframe>'';

</script>
</head>
<body>

<div id='ad1'>
<h5>Advertisement</h5>
</br>
<script language='javascript' type='text/javascript'>
postscribe('#ad1', b1);
</script>
</div>

<div id='ad2'>
<h5>Advertisement</h5>
</br>
<script language='javascript' type='text/javascript'>
postscribe('#ad2', b2);
</script>
</div>

<div id='ad3'>
<h5>Advertisement</h5>
</br>
<script language='javascript' type='text/javascript'>
postscribe('#ad3', b3);
</script>
</div>

<div id='ad4'>
<h5>Advertisement</h5>
</br>
<script language='javascript' type='text/javascript'>
postscribe('#ad4', b4);
</script>
</div>
</body>
</html>