

# SAS® 360 Match

## *Advanced Features Guide*

*Updated February 27, 2026*



# Contents

<b>Advanced Features</b> .....	<b>1</b>
<b>Count Directives</b> .....	<b>1</b>
<b>Action Tracking</b> .....	<b>2</b>
Data-Driven Attribution .....	3
<b>Viewability</b> .....	<b>4</b>
Implementing Viewability Support on Web Pages .....	4
Debugging Constants .....	4
Implementing Viewability Support in Creative Formats .....	4
<b>Lazy Loading</b> .....	<b>6</b>
Installation .....	6
Debugging .....	7
<b>Ad Passbacks</b> .....	<b>8</b>
<b>Advanced Tagging and Targeting</b> .....	<b>9</b>
<b>Supertags</b> .....	<b>9</b>
Namespacing .....	11
Supertag Use in Migrations .....	12
Supertag Use for Tag Management .....	12
Persisted Supertags .....	12
Token Substitution in Supertags .....	13
<b>Stacked Tags</b> .....	<b>13</b>
<b>Qualified Tags</b> .....	<b>15</b>
<b>BURST Tags</b> .....	<b>16</b>
<b>DISCOUNT Tag</b> .....	<b>17</b>
<b>Prefetch</b> .....	<b>17</b>
<b>Scheduling Beacon-Enabled Flights</b> .....	<b>18</b>
<b>Rotating IDs</b> .....	<b>18</b>
<b>Multivariant Testing</b> .....	<b>19</b>
<b>Define Radius</b> .....	<b>19</b>
<b>SLEEP Tag</b> .....	<b>20</b>
<b>Identifying Source of Email Prefetch Requests</b> .....	<b>20</b>
<b>Evaluating Multiple Data Sets</b> .....	<b>20</b>
ORDERBY Tag .....	21
<b>Handling Traffic Spikes</b> .....	<b>21</b>
BUSYNESS Tag .....	22

FIRSTLOOK Tag .....	22
REQPRIORITY Tag .....	22
<b>Single Sign-On: Setting Up with the Identity Provider .....</b>	<b>23</b>
Configure Single Sign-On with Okta.....	24
Setting Up Okta as IDP .....	24
IDP Data Required by SP .....	24
SP Data Required by IDP .....	24
Configuring SSO in SAS® 360 Match.....	25
<b>Data Activation .....</b>	<b>26</b>
Disabling Cookies .....	26
SETID .....	26
State Vector .....	27
State Vector Tags .....	27
Authenticating SETSV Calls.....	27
State Vector IDs.....	28
Secondary State Vector IDs .....	28
State Vector Events.....	29
Setting Tags via SETSV .....	30
Recording Events via SETSV .....	31
Explicitly Merging State Vectors via SETSV .....	32
Implicitly Merging State Vectors via SETSV.....	33
Loading Multiple Data Sets .....	33
<b>Bulk Data Upload via SV.....</b>	<b>33</b>
Data File Requirements .....	33
User Registration .....	34
Uploading Data Files.....	36
Duplicate Tags.....	37
Tokens .....	37
Deleting Personally Identifiable Information .....	37
<b>Identity Mapping.....</b>	<b>37</b>
<b>Data Expiration.....</b>	<b>38</b>
<b>Add UI Extensions .....</b>	<b>38</b>
<b>Create a Reverse Proxy .....</b>	<b>38</b>

## Advanced Features

SAS® 360 Match is a flexible advertising delivery platform able to serve online display, video, and mobile advertising channels. This document details some of the more advanced counting, tagging, targeting, and data manipulation features available in the software.

### Count Directives

Beacons assist in understanding and monitoring customer behaviors and activity and are a convenient way of gathering statistics. The use of a beacon allows a site to record the actions of a user opening the page that contains the beacon and allows the event to be counted. SAS® 360 Match supports several types of beacons that can be used in different contexts.

SAS® 360 Match has a generic event counting service that can deliver a response from a request based on the creative selected to be served. In SAS® 360 Match, actions are sent to the system using the count directive. For example:

```
http://{SASIA ad server URL}/{customer short name}/{contenttype}/count/  
fcid=1234/act=3/inc=1
```

increments the action count for flight creative ID, or FCID, 1234 [the unique creative identifier] by 1.

#### **act=1**

denotes a beacon count URL, or impression count. When the act value is missing or the act value is 0, the value defaults to 1 and the action defaults to an impression count.

Example:

```
http://shortname-ads.aimatch.com/{shortname}/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 about standard and custom action tracking, refer to Action Tracking in this document.

#### **act=4**

denotes a viewed beacon URL.

#### **act=5**

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

#### **inc=X**

increments the count by X. When the value for inc is missing, the value defaults to 1.

Other targeting tags present in the URL (for example, site or area) are logged against when the URL is executed. 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 FCID=57 and relocates to the click destination specified for that creative:

```
http://shortname-ads.aimatch.com/{shortname}/adclick/fcid=57/site=x/area=y
```

This example increments the click count for FCID=57 but does not relocate to the click destination:

```
http://shortname-ads.aimatch.com/{shortname}/count/act=2/fcid=57/site=x/area=y
```

## Action Tracking

**Standard action tracking** (for example, FCID, ADVID, ACTID) enables you to associate actions with a creative for the advertiser who owns the creative. The creative can be one that was recently seen or clicked by a visitor to the website. This feature is useful for monetizing behaviors such as leading a website visitor from an advertisement to a landing page where a transaction, such as a product purchase, subsequently occurs. Place the URL that tracks the action on the action success page, such as the landing or results page. When the visitor goes to the page, the URL can notify SAS® 360 Match 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 in the software.

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

```
http://{ad server domain}/{shortname}/count/act=3/fcid={FCID}
```

where {FCID} is replaced with the creative's FCID value. The code /act=3 tells SAS® 360 Match 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://{ad server domain}/{shortname}/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 in the software.

Site and area values are not needed in the action-tracking URL because they are captured at the time of the impression or during the 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://{ad server domain}/{shortname}/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 website visitor has viewed 50 percent of a video). Custom action tracking requires completion of the **Action Policies** section of the **Edit Flight** page in the software.

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 such cases, the custom action is logged even without a Custom Action Policy enabled for the flight. The active period for impressions or clicks is 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. However, if the policy has the **Allow Click to Trump Impression** option set (in Campaigns), 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 for the custom action to appear in the **Action Policies** drop-down menu in the **Edit Flight** screen in the software.

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 `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 in the software.

```
http://shortname-ads.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://shortname-ads.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 predefined actions that are available in the user interface and you can create additional custom actions in the **Custom Actions** section under the **Traffic** tab in the software.

## Data-Driven Attribution

When you use the `BYLABEL` keyword in a `COUNT` call, you can provide additional information to determine which ad's click or impression to attribute to an action. For example, you can use a combination of the `AVID` (advertiser ID) and the `SKU` for a creative to select the ad.

When SAS® 360 Match processes the call to determine which ad to attribute to an action, it uses the most recent ad from that advertiser that was served to or clicked by the visitor. The ad that is selected contains the creative with data that matches the value that is provided in the call. Matching is not case sensitive.

The additional creative data is entered into labels for the creative. For example, the following call matches a creative with a SKU label with the value of 2839-25292-B:

```
.../ADVID=32/BYLABEL/sku=2839-25292-b
```

The call can provide multiple values for the label to match. Multiple values are separated by commas. A creative with any of the values is a match. For example, the following call matches the creative above:

```
.../ADVID=32/BYLABEL/sku=2839-25292-a, 2839-25292-b, 2839-25292-c
```

A creative can contain multiple labels. The creative is considered a match if the COUNT call provides a matching value for any of its labels. For example, if a creative has a SKU label and a CATEGORY label with the value `blenders`, the following call is a match:

```
.../ADVID=32/BYLABEL/category=blenders
```

If the call provides multiple label and value pairs, a creative is a match if any of the values matches the creative's labels.

The action tracking remembers the last ad that is served for each flight. For SAS® 360 Match to remember multiple ads, each creative must be in a separate flight.

## Viewability

Viewability is a count of how many times a creative has been in view on a browser or device. Being in view is determined by how much of the creative's content is visible in the browser window, and how long it has been in view. Once those two thresholds have been met, the creative is considered viewed, and a viewed count is incremented for that flight creative.

Implementing viewability requires changes to the web pages where the viewability-enabled creatives are being served as well as changes to the creative formats that the creatives use. Viewability tracking works only with `jserver` and `bserver` ad call methods. To enable the viewability feature, contact SAS Technical Support at [support@sas.com](mailto:support@sas.com).

### Implementing Viewability Support on Web Pages

The following code snippet needs to be placed on every page where viewability tracking is desired:

```
<script type="text/javascript"src="https://content.aimatch.com/js/sasia/v1/sasia.min.js"></script>
```

### Debugging Constants

```
window.SASIA_VIEWTRACKER_DEBUG = true;  
window.SASIA_VIEWTRACKER_ALLOW_BLURRED = true;
```

### Implementing Viewability Support in Creative Formats

All system creative formats already contain the necessary code to implement viewability. The creative format code is not used unless the viewability code snippet above has been

implemented on the page, Custom creative formats need to be modified to support viewability. Specifically, attributes and tokens need to be applied to the HTML elements displaying the actual creative content. These are:

```
id="sasia-fcid-%%FCID%%"  
  class=" sasia fcid"  
  data-sasia-view-time="%%VIEWABILITYTIME%%"  
  data-sasia-view-percent="%%VIEWABILITYPERCENT%%"  
  data-sasia-view-url="%%VIEWABILITYURL%%"
```

## IMG Elements

Add the attributes directly to the IMG HTML tag:

```
<a href="%%CLICKURL%%" target="_blank"></a>
```

## IFRAME Elements

Add the attributes directly to the IFRAME HTML tag:

```
<iframe src="%%MEDIA%%?clickTag=%%PRECLICKURL%%" width="%%WIDTH%%"  
height="%%HEIGHT%%" scrolling="no" allowtransparency="true" marginwidth="0"  
marginheight="0" vspace="0" hspace="0" noresize="true" frameborder="0"  
align="left" style="border:0px none;padding: 0px ;margin:0px; float:none;"  
id="sasia-fcid-%%FCID%%" class="_sasia_fcid" data-sasia-view-  
time="%%VIEWABILITYTIME%%" data-sasia-view-percent="%%VIEWABILITYPERCENT%%" data-  
sasia-view-url="%%VIEWABILITYURL%%"></iframe>
```

## SCRIPT Elements

Surround the JavaScript or SCRIPT HTML tag with a span tag containing the attributes:

```
<span id="sasia-fcid-%%FCID%%" class="_sasia_fcid" data-sasia-view-  
time="%%VIEWABILITYTIME%%" data-sasia-view-percent="%%VIEWABILITYPERCENT%%" data-  
sasia-view-url="%%VIEWABILITYURL%%">  
<script src="%%X_SCRIPT_SRC_URL%%&ncu=%%PRECLICKURL%%&ord=%%RANDOM%%">  
</script>  
</span>
```

Do the same for JavaScript that uses document.write() to construct a SCRIPT tag:

```
<span id="sasia-fcid-%%FCID%%" class="sasia_fcid" data-sasia-view-  
time="%%VIEWABILITYTIME%%"  
data-sasia-view-percent="%%VIEWABILITYPERCENT%%" data-sasia-view-  
url="%%VIEWABILITYURL%%">  
<script type="text/javascript">  
  var type = "%%X_AD_TYPES%%";  
  var skySource = "m=3&tp=7&d=j&t=n";  
  var bannerSource = "m=1&tp=5&d=j&t=n";  
  var rectSource = "m=6&tp=8&d=j&t=n";  
  if (type == "sky") { _adTypeSource = skySource; }  
  if (type == "banner") { _adTypeSource = bannerSource; }
```

```
        if (type == "rect") { _adTypeSource = rectSource; }
        document.write('<scr' + 'ipt language="javascript"
src="http://media.example.net/w/get.media?sid=%X_SID%%&' + _adTypeSource +
'&walsh=%PRECLICKURL%%"></scr' + 'ipt>');
</script>
</span>
```

## Lazy Loading

When properly instrumented, ads rendered outside the browser viewport are not loaded on the initial document load. Only when ads are in or near the viewport is the actual ad call made.

### Installation

Install the `sasia.js` library on your page:

```
<script src="https://content.aimatch.com/js/sasia/v1/sasia.min.js"></script>
```

To use lazy loading, you must also include the `postscribe` scripts separately. These two **must** be loaded before `sasia.js` on the page.

```
<script src="/path/to/your/htmlParser.js"></script>
<script src="/path/to/your/postscribe.min.js"></script>
```

If `postscribe` is not available on the page before SAS<sup>®</sup> 360 Match is included, the lazy loading functionality is not initialized. For more information about `postscribe`, refer to <https://github.com/krux/postscribe>.

Once initialized, the lazy loader starts scanning the page for ad calls that are not loaded. It loads ads that are in the viewport or within 200 pixels (default) of the viewport. Set your own distance threshold after the script has been loaded using the `setDistanceThreshold` method.

```
SASIA.LazyLoader.setDistanceThreshold(100);
```

To enable individual ad calls to be lazy loaded, the tags must be instrumented so that the `LazyLoader` can detect and load properly. You do this with a `div` or `span` tag that has a `sasia-lazy-ad` class name. A tag with the `sasia-lazy-ad` must also have a `data-lazy-ad-src` attribute containing a JavaScript src ad call URL. For example:

```
<span class="sasia-lazy-ad" data-lazy-js="/path/to/js/ad/call"> </span>
```

Define the ad space dimensions in advance to enable the page to be rendered correctly the first time and to not trigger another rendering of the page layout when the ad loads. In this way, when the ad loads lazily after the page has already finished loading, the layout of the page already has space for the ad. For example:

```

<style>
  .ad468x80 {
    display: inline-block;
    width: 468px;
    height: 80px;
  }
</style>
<span class="_sasia-lazy-ad ad468x80" data-lazy-
js="/path/to/your/468x80/js/ad/call/">

```

Lazy loading and view tracking will work together if the ad calls are tagged in this way, and if the rendered ad call text, after lazy loading, have the correct attributes on the tags to be detected. See the Viewability section for more information.

## Debugging

There are variables that you can declare to cause SAS® 360 Match to emit some verbose debugging information to the console. This is not recommended in production but can be helpful in getting it set up correctly. Be sure to turn debugging off when it is verified to be working as expected.

To enable debug info for view tracking, declare this anywhere on the page:

```

window.SASIA_VIEWTRACKER_DEBUG = true;

```

To enable debug info for the lazy ad loading, declare this anywhere on the page:

```

window.SASIA_LAZYLOADER_DEBUG = true;

```

You can enable one or both types of debugging. For example:

```

<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Lazy Loading and View Tracking Example</title>

  <style>
    .ad300x250 {
      display: inline-block;
      width: 300px;
      height: 200px;
    }

    .above-the-fold {
      width: 600px;
      height: 800px;
      padding: 24px;
      margin-bottom: 24px;
      border: 2px solid #0099ff;
      background-color: #ffffe6;
      color: #0099ff;
    }
  </style>

  <script type="text/javascript">
    window.SASIA_VIEWTRACKER_DEBUG = true;

```

```

        window.SASIA_LAZYLOADER_DEBUG = true;
    </script>

    <script src="/path/to/your/htmlParser.js"
type="text/javascript"></script>
    <script src="/path/to/your/postscribe.min.js"
type="text/javascript"></script>
    <script src="//content.aimatch.com/js/sasia/v1/sasia.min.js"
type="text/javascript"></script>

    <script type="text/javascript">
        SASIA.LazyLoader.setDistanceThreshold(100);
    </script>
</head>
<body>

<h1>Lazy Loading and View Tracking Example</h1>

<div class='above-the-fold'>
    <h4>This is a large block to push the ad call down the page to
demonstrate lazy loading</h4>
</div>

<h2> The Ad Call</h2>
<span class="_sasia-lazy-ad ad300x250" data-lazy-ad-
src="/javascripts/ad-call-img.js"> </span>
</body>
</html>

```

## Ad Passbacks

During normal ad selection, the engine remembers the path information, the query string, and blocked advertisers of the last 10 ads that were served in the last five seconds, for each visitor. The number of ads that are remembered for each customer can be configured by SAS Technical Support. 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 typically triggered by a third party that redirects the request to SAS® 360 Match 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 in the software.

The SASPB and FCID passback tags that are sent by the third party are ideally supplied by a template that creates them in response to the original ad served. Otherwise, they must be hardcoded in the trafficking of SAS® 360 Match passbacks by the third party itself, usually by size.

If the passback FCID is for a valid ad recently served to the visitor, its associated path information, query string, and blocked advertisers are resurrected for the passback ad selection. The ad's advertiser is added to the blocked advertisers for this ad selection only. Ad selection starts at the top of the queue and blocks selecting any flight associated with the blocked 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 generate the same set of tags for the ad because only the most recent details are 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 based on the ad's timestamp, tags, and other information. This happens only for ads that are in a 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. A passback request can be modified to change the tag values from the original request or add new tag values. For example,

```
Original request: /tagA=1/tagB=2/tagC=3/tagD=4  
Passback: /tagD=27/tagE=5  
Result: /tagA=1/tagB=2/tagC=3/tagD=27/tagE=5
```

The passback request is logged and the count can be found in the Usage Report.

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

## Advanced Tagging and Targeting

SAS® 360 Match provides many ways to create, manage, and personalize digital content and settings.

### 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 ad servers and provide a level of tag management to ad operations. Supertags provide a way of mapping a value to an arbitrary path info snippet. You can create a maximum of 1,000 supertags. Contact SAS Technical Support if you want to change this limit.

For example, a supertag value `HOMETOP` could be defined as follows:

```
AREA=HOME/POSITION=TOP/SIZE=LEADERBOARD
```

Then the following ad call:

```
http://shortname-ads.aimatch.com/customer/hserver/supertag=hometop
```

would be equivalent to this ad call:

```
http://shortname-ads.aimatch.com/customer/hserver/area=home/position=top/size=leaderboard
```

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 path info 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 did not 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, where the most recent definitions take 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://shortname-ads.aimatch.com/customer/hserver/supertag=hometop,localnews
```

would be equivalent to this:

```
http://shortname-ads.aimatch.com/customer/hserver/position=top/size=leaderboard/site=news/area=local
```

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

```
http://shortname-ads.aimatch.com/customer/hserver/supertag=localnews,hometop
```

would be equivalent to this:

```
http://shortname-ads.aimatch.com/customer/hserver/site=news/position=top/size=leaderboard/area=home
```

An ad call with a supertag value can include other tags as well. The complete set of tags implied by that mix determines what ad is served. When a supertag value maps to path information that includes a tag that is included in the ad call, the explicit value from the ad call takes precedence. Using the `HOMETOP` example above, the following ad call:

```
http://shortname-ads.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 path info 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. For example, the path info definition cannot be used to alter the number of ads returned by a bserver call.

The path info 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 path info 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.

Supertag values are not case sensitive. They can contain all special characters except for pipes and commas. Special characters should be properly encoded to be included in a URL. 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 path info. 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 does not 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/KIDS=NO/ THEME=ADULT`. First, it expands to `/NAME=THISISAPROGRAM/GENRE=MA`, and then `GENRE=MA` gets expanded to `/KIDS=NO/THEME=ADULT`. 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 the back end, the 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 SAS<sup>®</sup> 360 Match for each slot known by the previous system. The supertag value is set to the same name as the slot, and the path info is set to the equivalent set of tags and values in SAS<sup>®</sup> 360 Match. Most commonly, a slot maps to a specific site, area, and size. When retagging pages for SAS<sup>®</sup> 360 Match, you can exchange references to your former server's slot names for SAS<sup>®</sup> 360 Match ad calls referencing SUPERTAG. In some cases, you can use JavaScript supplied by SAS to reference those names.

## Supertag Use for Tag Management

Supertags also give you a means to change information in your ad calls without changing code on your web pages. For example, you can assign each unique ad and 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 path info needed for each supertag value, and then change these freely over time.

## Persisted Supertags

Use persisted supertags to load and use data within a single ad request. Persisted supertag data is uploaded to SAS<sup>®</sup> 360 Match using a data file similar to user registration data. Unlike user registration data, however, persisted supertag data does not remain in the visitor's session once the data is loaded. Instead, persisted supertags are used only within a single ad request evaluation like traditional supertags.

Like traditional supertags, data from a persisted supertag can be invoked directly by including `IDNAME=IDVALUE` in the ad call, or indirectly through other supertags (persisted or traditional), other user registration data, or state vector data. The data activated by persisted supertags can further invoke data for other supertags.

Here are the differences between persisted and traditional supertags:

- Persisted supertag data is provided through data uploads. Traditional supertags are defined through the SAS<sup>®</sup> 360 Match interface or its XML API.

- Persisted supertags can be invoked only using the `IDNAME=IDVALUE` syntax. Only one value can be supplied for `IDVALUE`. If multiple values are provided, only the first one is used.

Persisted supertags must have an ID column with a unique name, and one or more other columns representing tag data to be loaded when that ID is referenced. For `bserver` or `bserverj` calls, different values for the ID can be defined to represent different data like traditional supertags.

For example, consider the following data upload for a persisted supertag with an ID name of `PRODUCT_ID`:

```
PRODUCT_ID,CATEGORY,DESCRIPTION
12982,Household,Waring Blender
22829,Sports,Titleist Golf Balls
```

An ad call that includes `PRODUCT_ID=22829` adds `CATEGORY=Sports/DESCRIPTION=Titleist Golf Balls` to that request, and only that request. If the next request provides no value for `PRODUCT_ID`, then `CATEGORY` and `DESCRIPTION` are empty for that request. This is the key difference between persisted supertags and user registration data.

For a `bserver` call with `/B1/PRODUCT_ID=22829/B2/POS=TOP/B3/PRODUCT_ID=12982/`, the first ad is evaluated with `CATEGORY=Sports/DESCRIPTION=Titleist Golf Balls`, the second with just `POS=TOP`, and the third with `CATEGORY=Household/DESCRIPTION=Waring Blender`.

Contact SAS Technical Support to configure persisted supertags.

## Token Substitution in Supertags

You can use tokens to substitute the values in path information for supertags. Use the substitution to build tag values dynamically using existing formatting tokens, such as `SPLIT` and `REGEX`.

For example, you can use a targeted supertag to transform a tag on request into another tag with a different value.

You can define a targeted supertag for a request with `/tagA=abcd_0001` with the following specifications:

```
Target: tagA IS NOT EMPTY
Path information: /tagB=%%tagA:REGEX,(*)_d+$,1%%
```

In this example, `tagB` contains the value of `abcd`. This rule eliminates the need to create a matching supertag for every value of `tagA`. For more information about using tokens, see [Using Tokens](#) in the SAS 360 Match User's Guide.

## Stacked Tags

SAS® 360 Match supports stacked tag values. A stacked tag has multiple values that are taken one at a time in subsequent ad requests. Once the tag is set in 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.

The stacked tags cycle indefinitely unless the visitor has a view ID. For each visitor, if the same view ID is sent in multiple requests, each ad request processes the next value until all the values in the stack have been shown for that view ID. After using the last value, subsequent ad requests use an empty value.

If the view ID changes for the visitor, the ad request starts to evaluate the next value in the stack until all the values have been shown for the new view ID. This might mean starting to show a value from the middle of the stack and cycling back to the top.

In addition, stacked tags that are invoked by a tag qualifier only advance if the creative supplying the qualifier value is the one that is chosen.

In the following examples, the visitor and the view ID do not change.

Any tag can be given stacked values by prefixing the values with "!" (exclamation mark) and separating them with more "!"s. For example:

```
offer=!beer!soda!nuts
```

results in the following:

```
offer=beer in the first request,  
offer=soda in the second request,  
offer=nuts in the third request, and  
offer=<empty> in subsequent requests.
```

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:

```
offer=!beer,soda!nuts
```

results in the following:

```
offer=beer,soda in the first request,  
offer=nuts in the second request, and  
offer=<empty> in subsequent requests.
```

An empty position in the stack can be expressed with multiple "!". For example:

```
offer=!beer!!soda
```

results in the following:

```
offer=beer in the first request,  
offer=<empty> in the second request,  
offer=soda in the third request, and  
offer=<empty> in subsequent requests.
```

Multiple tags can be given stacked values and they advance through their stacks together. For example:

```
offer=!beer!soda!nuts/pos=!1!2!3
```

results in the following:

```
offer=beer/pos=1 in the first request,  
offer=soda/pos=2 in the second request,
```

```
offer=nuts/pos=3 in the third request, and  
offer=<empty>/pos=<empty> in subsequent requests.
```

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 user registration, `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 like an explicit value can temporarily override a geo lookup. The override does not affect the stack. If the next ad request does not include the tag, the next stacked value is used.

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

The beacon (vericount) URL and click URL for an ad 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 was served using a stack value, the new ad served uses the same value for that tag and does not disturb the stack.

If a `VIEWID` remains constant throughout an entire cycle, the cycle stops and the tag is treated as if it has no value. When the `VIEWID` changes, the cycle resumes from the last used position in the stack. If no `VIEWID` is present, the values cycle indefinitely.

It is possible to provide recommendations when a specific key is encountered. A visitor's positions in multiple sets of stacked tags can be maintained. When a set of tags is read using the ID (for example, product ID), they are kept with the visitor under a `PRODUCT_ID.VALUE` qualification for the balance of the session. When the ID changes, if the visitor already has tags with that new value's qualification, they are used, rather than re-reading from the data store. And if they contain any stacked tag context, they resume where they left off.

The data import is the same as other ID-based data lookups. There must be a header row defining columns for the ID and each of the tags with associated data. For example:

```
PRODUCT_ID,RECOMMENDATIONS,OTHERTAG  
54321,!2352!3262!6325,SOMEVALUE  
54322,!3523!6262!8629,ANOTHERVALUE
```

## Qualified Tags

Qualified tags allow a tag to have any number of independent sets of values, selectable via a qualifier during ad evaluation. A tag qualifier column in the flight creatives table determines which qualified tags are temporarily switched into the evaluation context while evaluating that creative for serving. A check box for qualified tags appears in the SAS® 360 Match user interface when editing a flight creative in an active flight.

A qualified tag is expressed by prefixing the tag name with a qualifier string and dot (for example, `CR18.SKU=!100!150!180` expresses three stacked values for a SKU tag qualified by `CR18`). The qualifier can be a string containing any characters that do not interfere with parsing path info contents (for example, `/ = ? &`). The qualifier itself can include a dot. The last dotted

portion of the qualified name is assumed to be the base tag name (for example, CR18.CTL.SKU has CR18.CTL as a qualifier for the SKU tag). Case is ignored.

Qualified tag values can be supplied in the same contexts as regular, unqualified tags, including:

- Ad call path info
- SETSV, including being able to append or expire values
- User registration

A qualified tag can accept the same types of values as a regular tag (stacked, multivalued, and so on).

Qualified tags are ignored during ad serving unless a creative specifically requests a qualification. When a flight creative with a tag qualifier is evaluated, all of the tags defined with that qualifier become the active values for the base tag when the creative is evaluated. The qualified tag values are used during evaluation of the creative's targeting, and in any token substitutions in network creatives.

For example, consider a visitor with the following tags set:

```
.../GENDER=M/CR18.SKU=123/CR19.SKU=124/CR34.SKU=89/CR19.PROD=2868/...
```

When evaluating a creative with `tag_qualifier="CR19"`, the engine treats the visitor as logically having these tags:

```
.../GENDER=M/SKU=124/PROD=2868/...
```

When evaluating a creative with `tag_qualifier="CR34"`, the engine treats the visitor as logically having just:

```
.../GENDER=M/SKU=89/...
```

because there is no qualifying value for the `PROD` tag.

If a creative with a tag qualifier is ultimately selected to serve, then all summary logging and sampling includes the values of any qualified tags involved in the decision. Also, all such tag values are sent back to the direct server, or dserver, to be use in token substitution. A qualified tag with stacked values is advanced to the next value in the stack only when the creative served is associated with the tag's qualifier.

**Note:** By default, a dserver request returns a maximum of 100 creatives. To increase this value, contact SAS Technical Support.

## BURST Tags

Use the BURST tag in an ad request to allow a flight to act as top priority until a specific goal is met. Before the flight goal is achieved, a flight with this tag can serve without being paced. After the goal is achieved or exceeded, the flight follows its own pace settings.

The BURST tag has an integer value between 0 and 100, which represents the percentage of the flight goal. Before a flight reaches the percentage of its goal, it is treated as top priority during the ad request. After a flight achieves that percentage of its goal, its normal pacing behavior is respected.

For example, consider Flight A that has served 63% of its goal and Flight B that has served 82% of its goal. Both flights are in tiers that use schedule priority and both are slightly ahead of schedule. Neither flight is set to top priority.

Consider several ad requests with different values for the BURST tag:

BURST Tag	Flight A	Flight B	Comment
No BURST tag or BURST=0	Cannot serve	Cannot serve	Both flights have served more than 0% of their goals. Because they are both ahead of schedule, neither can serve.
BURST=60	Cannot serve	Cannot serve	Both flights have served more than 60% of their goals. Because they are both ahead of schedule, neither can serve.
BURST=80	Can serve	Cannot serve	Flight A has served less than 80% of its goal. Therefore, the flight's pacing is ignored, it is treated as top priority, and the flight can serve. Flight B has served more than 80% of its goal, so it follows its pace settings.
BURST=85	Can serve	Can serve	Both flights have served less than 85% of their goals. Therefore, pace settings are ignored for both, they are treated as top priority, and the flights can serve.

The value for percent of goal served is based on served impressions. The flight's delivery patience, on-schedule percentage values, or current calculated priority do not affect how the BURST tag is used. However, these values are considered when a flight is not subject to a BURST override and must respect its pacing.

## DISCOUNT Tag

Use a DISCOUNT tag for a flight that has a value goal or a daily revenue cap to dynamically alter the amount that is credited to the goal when an event, such as an impression, click, view, or action, occurs.

The value for the DISCOUNT tag is a decimal number that represents the proportion of the value to subtract from the nominal value. A negative discount marks up the value.

For example, a flight has a cost per thousand (CPM) impressions of 3.00 and a DISCOUNT=0.10 tag is included in an ad request. An impression from the flight in the call is treated as if the CPM is 2.70, which is a 10% discount.

A best practice is to use targeted supertags instead of including the DISCOUNT tag in an ad call. Using supertags enables you to control how the discount is applied within SAS 360 Match, rather than giving control to the publisher. For example, you can create a supertag that automatically applies DISCOUNT=0.25 when a target that defines the discounted inventory matches. You can change the discount amount or the supertag target at any time without requiring changes to the publisher's website.

## Prefetch

When an ad is selected, targeting for time, day of week, or the part of a day (such as start and end times for a flight) is evaluated based on when the request was received.

However, when an ad is selected but not rendered until a later, unspecified time, the ad might appear outside the time frame of the targeting. This could cause a prefetched ad with restricted content to serve outside approved hours. A prefetched ad could also serve outside contracted times.

Use the PREFETCH tag in the ad call to specify a future time when a prefetched ad is expected to be rendered. The prefetch value is used for targeting.

The value for the PREFETCH tag can be absolute or relative. When the value uses the YYYY-MM-DDTHH:MM:SS format, it represents an absolute value for UTC. In this case, HH must be between 00 and 23. In addition, use a T to separate the date and time, such as 2021-12-23T14:12:30. When the value is a single number, it represents a relative number of seconds in the future. If the value is in neither of these formats or if the value implies a time in the past, the PREFETCH tag is ignored and the value is assumed to be 0.

BSERVER and BSERVERJ calls can contain separate prefetch times in each B section. If the prefetch time is expressed in the BALL section, the same prefetch time is used for all ads.

For DSERVER calls, the same prefetch time is used for all ads.

To suppress the prefetch behavior, even in the presence of the supertag lookup that produces a prefetch value, include /PREFETCH=0/ in the call to override any value that is produced by the supertag lookup.

## **Scheduling Beacon-Enabled Flights**

You can control how to schedule flights with beacon-based goals. By default, the pace of a beacon-enabled flight is based on the beacons that arrive. However, you can enable a setting that determines the schedule of a flight based on the sum of the number of beacons that are received and a short-term estimate of the number of future beacons.

The estimated number of future beacons is based on a ratio of the number of beacons received and the number of flight selections in the past one to two hours. The ratio is updated at regular intervals as new flight selections are made.

If the flight has no selections or beacons in the past one to two hours, the count of actual incoming beacons is used for pacing.

Contact SAS Technical Support to enable this feature. This setting affects flights with view, click, or action goals. In addition, this feature affects flights with impression goals that have beacon counting enabled.

## **Rotating IDs**

Use the ROTATINGID token to generate universally unique IDs (UUIDs) for users that changes each month, based on the user's time zone.

The rotating UUID is generated at the start of a visitor session. It is a combination of the user's MID value and the current month. For registered users who are using different devices, the UUID is consistent across devices because the DM value is consistent across the devices.

The same UUID is always produced for the same user when the session begins in the same month. When the current time rolls into a new month, the user gets a new UUID. However, the UUID remains constant for the duration of a session, even if the current time rolls over to the next month. The UUID is logged in sampling files to aid in debugging.

By default, SAS® 360 Match does not produce rotating IDs. Contact SAS Technical Support to enable this feature.

## Multivariant Testing

You can enable multivariant support in SAS® 360 Match by using either a multivariant test tier or by setting the sticky duration of a flight. This setting causes the decision engine to remember which creative it has served and to continue to serve that creative for a period that is determined by the value of the sticky duration for the flight or test tier.

When the sticky duration is configured and saved for a flight, the creative that is selected for the flight remains sticky for the specified duration. In a test tier, the sticky value of 0 indicates that a selection remains sticky forever.

Once a creative is chosen, it continues to be chosen if the creative's size and targeting requirements match. If they do not match, SAS® 360 Match attempts to select another creative from the same flight. If another creative from the same flight is found, it becomes sticky too. When this flight is evaluated in subsequent requests, all the sticky creatives are considered, according to the flight's creative selection method.

The sticky duration is set for each visitor. When a creative is selected from a flight, it remains sticky for that visitor. Every time that the flight is examined for the visitor, the same creative is checked first to determine whether it can serve. If the creative matches the request (and no other creative in the flight does), the visitor is always served that creative when that flight serves.

For a flight in a multivariant tier or with a sticky duration, you can configure a sticky set name on a flight creative. When such a creative is selected, all other flight creatives that have the same set name also become sticky and are considered in subsequent requests. If none of the sticky creatives match in a subsequent request, SAS® 360 Match attempts to select another creative from the same flight. If another creative from the same flight is found, it and any other creatives in its set also become sticky.

When a flight creative that has a set name is served, the FCSET token becomes available for token substitution in the resulting creative content. Its value is that of the flight creative's set name.

## Define Radius

SAS® 360 Match enables you to specify a radius around a session's last known location. If any activity originates outside of the radius, the session is assumed to be a bot. Any subsequent activity does not count for the duration of the session.

The radius is determined by comparing the locations of the most recent request and the current request. If the distance between the two requests is greater than the maximum radius, the session is treated as a bot.

Contact SAS technical support to configure a maximum value for the radius. By default, no maximum radius is specified.

## SLEEP Tag

Use the SLEEP tag to simulate slow ad responses. Add the /SLEEP=<ms value> to the ad request. The length of the pause is specified in milliseconds. This feature does not affect your customers.

By default, you must provide a valid API key to use the pause feature. Contact SAS technical support to disable the authentication requirement.

## Identifying Source of Email Prefetch Requests

You can use geotargeting values to identify the source of email prefetch requests.

- A Mail Privacy Protection request from Apple is identified by GEO\_PROXY\_TYPE = “consumer-privacy” and GEO\_PROXY\_DESCRIPTION = “apple”.
- A Google Gmail prefetch request is identified by GEO\_PROXY\_TYPE = “consumer-privacy” and GEO\_PROXY\_DESCRIPTION = “google”.

To avoid counting views of one of these types of requests, use a targeted supertag to add /nolog for a specific proxy type and description.

## Evaluating Multiple Data Sets

You can configure a list of supertag names that can generate multiple data sets from multiple values. This enables you to define the relationship between an item and a list of other identifiers with independent data and overlapping tags.

For example, a visitor has many of item X and each item of type X has a COLOR tag with a different value. You can configure SAS® 360 Match to make targeting decisions based on the distinct sets of data – in this case item X and COLOR – to find the flight and creative to serve.

When the ad request is received, SAS® 360 Match checks tags from the request pathinfo, visitor state vectors, and other supertag expansions to find any tags from the configured supertag list. The tags in the configured list are searched in the order that they are configured.

If one of the tags is present in the current tag data and contains multiple values, each value produces a new tag data set. By default, the data sets are evaluated in the same order as the values that produced them. For more information about ordering data sets, see the section on the ORDERBY tag.

The tags can be nested. A data set can invoke supertags that generate more data sets that inherit from their parent data set. The targeted supertags are evaluated separately and in the order that the tag values that spawned them from the supertags.

The tag expansion process generates one or more complete tag data sets that Match uses during ad selection.

Contact SAS technical support to configure the ability to evaluate multiple data sets.

## ORDERBY Tag

Use the ORDERBY tag to specify how to order multiple data sets for an ad request.

The ORDERBY tag lists the tags for the sort. The first tag is the primary sort tag, the second tag is the secondary sort tag, and so on. By default, the sort order is ascending. To change the sort to descending order, add the “-“ prefix to the tag name.

If all tag values are numeric, a numeric sort is done. Empty values appear first in an ascending sort and last in a descending sort. If any non-empty value is non-numeric, an alphabetic sort is done.

Use the YYYY-MM-DD format to sort by date.

When there are equal values in the data sets, the data sets are added in the order that they are listed.

If no ORDERBY tag is present, then the data set order remains determined by the order of the tag values that created them.

For example, consider the following data sets that would ordinarily be ordered like this:

```
D1: /ACCT=1/OFFER=123/APR=2.3/EXPDATE=2025-02-28
D2: /ACCT=1/OFFER=789/APR=3.4/EXPDATE=2025-02-15
D3: /ACCT=2/OFFER=483/APR=2.1/EXPDATE=2025-02-15
D4: /ACCT=2/OFFER=286/APR=4.5/EXPDATE=2025-03-15
```

The table shows how the data sets are ordered:

ORDERBY Value	Data Set Order	Explanation
No tag	D1, D2, D3, D4	No ORDERBY tag is present. The data set order is determined by the order of the tag values that created them.
=APR	D3, D1, D2, D4	The data sets are sorted by ascending order based on the value of APR.
=-APR	D4, D2, D1, D3	The data sets are sorted by descending order based on the value of APR.
=EXPDATE	D2, D3, D1, D4	The data sets are sorted by ascending order based on the value of EXPDATE. D3 and D3 have the same value for EXPDATE and are listed according to the tag values that created them.

After the ORDERBY tag is applied and your data sets are ordered, SAS 360 Match applies the data set limit. The application discards any data sets that exceed the maximum number of allowed data sets. A recommended practice is to manage data set keys such that unnecessary or invalid data sets do not add unnecessary decisioning latency.

## Handling Traffic Spikes

While SAS 360 Match automatically scales for traffic changes, the system also provides several methods to manage unplanned traffic spikes and prevent service errors and time outs in your network.

NOTE: These features are available for Match Premium subscribers. All customers of SAS 360 Match benefit from the automatic scaling of the ad server to handle increases in traffic.

## **BUSYNESS Tag**

Use the BUSYNESS tag to override normal decisioning during sudden traffic spikes. The BUSYNESS tag can be combined with a supertag to direct traffic to the FIRSTLOOK tier. For example, use a supertag targeted to BUSYNESS  $\geq 3$  with a value of FIRSTLOOK=busytier.

## **FIRSTLOOK Tag**

Use FIRSTLOOK=<tier name> in the impression request to specify that the ad server should check for a matching flight in the specified tier first. The tag enables you to skip regular ad selection and replace it with an ad from the FIRSTLOOK tier.

The FIRSTLOOK tag can be in the request itself or included in limited supertags. Only one tier can be named in the FIRSTLOOK tag. If multiple tiers are listed, only the first tier is used.

SAS 360 Match selects flights from the FIRSTLOOK tier only. No visitor, geographic, or device data are available for targeting. The only information is provided directly in the request or synthesized by supertags. Normal ad selection logic applies. However, the flights and their campaigns, creatives, and formats cannot use any business logic that requires visitor-specific information, such as exposure policies.

If a non-default ad is selected from the FIRSTLOOK tier, it is served. For a BSERVER or DSERVER request, if a non-default ad is selected for the first ad, the whole bundle of ads is served. However, no bot or fraud detection is performed. Flights in a FIRSTLOOK tier should consist only of house or special ads with serve counts that do not affect revenue.

If a default ad is selected from the FIRSTLOOK tier (or, in the case of a BSERVER or DSERVER request, the first ad is the default), or if the FIRSTLOOK tier does not exist, a normal ad selection is performed.

When an ad call that contains the FIRSTLOOK tag is traced using TRACE=1, the initial portion of the trace shows the FIRSTLOOK tier that was named. There is no visitor data. If the FIRSTLOOK selection produces a default ad, the initial trace result is followed by a second trace that shows the regular ad selection that was performed.

## **REQPRIORITY Tag**

By default, ad requests are handled on a first-come, first-served basis. Optionally, COUNT and SETSV requests can be configured to be a lower priority so that these requests are processed later than other requests.

When faced with a traffic spike, requests may take longer to complete, and clients may time out while waiting for replies. SAS 360 Match Premium subscribers could benefit from using the REQPRIORITY tag to specify the relative importance of specific requests to each other.

Use the REQPRIORITY tag in an ad request or in supertags to modify the priority of certain requests and specify that these requests are handled first when the network is busy. This is a form of load shedding in ad decisioning, placing control over the requests of lesser value in the hands of the customer.

The following are the values for the REQ PRIORITY tag:

1: A high-priority request that is processed on a first-come, first-served basis. This is the default value if no REQ PRIORITY tag is provided.

2: A mid-priority request that is processed only after all high-priority requests (REQ PRIORITY=1) are completed.

3: A low-priority request that is processed only after all high- and mid-priority requests (REQ PRIORITY is 1 and 2) are completed.

COUNT and SETSV requests are processed after all other requests have been completed. If the BufferSetSV and BufferBeacons customer setting is enabled, or the customer also prioritizes COUNT and SETSV requests using the REQ PRIORITY tag.

SETSV and SET requests cannot manipulate the engine REQ PRIORITY tag data. If the REQ PRIORITY tag is used as a supertag, use VTT to manipulate the engine data.

## Single Sign-On: Setting Up with the Identity Provider

Single sign-on, also known as SSO, provides a way of authenticating users in SAS® 360 Match through a third-party identity provider, or IDP. The communication between SAS® 360 Match and the IDP uses Security Assertion Markup Language, or SAML. To configure single sign-on, you must set up the IDP and the service provider, or SP, to exchange required data. In this case, the SP is SAS® 360 Match.

Here are the terms used for the SSO process:

### Identity Provider (IDP)

Provides identities for users who need to interact with a system. The IDP is usually the host of the user information repository that handles authentication and password management.

### Security Assertion Markup Language (SAML)

Enables "single sign on" by allowing a single-user authorization service (or identity provider) to grant access to any number of third-party or remote applications. For example, when SAML is set up and configured, the identity provider can provide credentials to log on to applications such as Airbrake, PagerDuty, or Slack.

### Service Provider (SP)

Provides features or functionality to users. The IDP, rather than the SP, manages user authentication.

### User Agent

Usually refers to a web browser.

## Configure Single Sign-On with Okta

To configure single sign-on with, you must set up the Identity Provider (IDP) (Okta) and the Service Provider (SP) (that is, SAS® 360 Match) to exchange required data.

### Setting Up Okta as IDP

After logging in to Okta, select **Add Applications** in the **Shortcut** menu on the dashboard. Click **Create New App**. Then click **Show Advanced Settings** to display all the fields in the **SAML Settings** panel.

### IDP Data Required by SP

The SP must receive the following information from the IDP:

- the "IDP single sign on" URL to redirect users for authentication
- the IPD issuer URL
- a copy of the IDP X.509 certificate

These values can be entered manually or provided by a metadata XML file to the SP.

Also, the Security Assertion Markup Language (SAML) uses the **Name ID Format** value to enable the IDP and SP to exchange information about a user, and to identify the user. SAS® 360 Match expects the Name ID to be **Persistent**. (The Name ID can be configured to have a transient value).

### SP Data Required by IDP

The IDP must receive the following information from the SP:

- The SP URL to post authentication information, that is, the SP "consumer" or "SP single sign on" URL.
- The default SP landing page, that is, the "Audience URI" or "SP Entity ID". This is typically the home page for the website.
- Authentication user payload information that is sent to the IDP in the format required by the SP. An SP defines the user attributes (that is, "Claims") that are acceptable. For example, the attributes might be "login" instead of "username", "first\_name" instead of "FirstName", or "email" instead of "email\_address". If necessary, the IDP transforms its user data to the expected format before posting it to the SP consumer URL.

**Note:** SAS® 360 Match expects to receive the following attributes from the IDP (case sensitive)

- FirstName
- LastName

- Email
- Login
- SAS® 360 Match requires the four attributes and the persistent name ID from the IDP when posting to the SSO consumer.
- (Optional) Assertion encryption certificated generated by SAS® 360 Match. Configure the IDP to accept an encryption algorithm of AES-256-CBC and key transport algorithm of RSA-OAEP.

## Configuring SSO in SAS® 360 Match

**Note:** The IDP configuration requires information from the SP, and the SP requires information from the IDP during setup. You might want to create an initial SSO configuration with only name and default role set and leave the URL fields blank before you configure the IDP. Configuring the SSO in this manner enables you to have the required information before you get started.

The fields in the **SSOs** panel are automatically populated after metadata information is copied into the **Metadata** field that appears when you first select the **SSOs** panel on the **Admin** tab.

During the initial setup, do not set the SSO configuration as the default until it is proven to work. Once the SSO configuration is set to be the default, navigating to the URL of the user interface will redirect you to the IDP's logon screen, if the user is not logged in.

After logging in through the IDP, a new user account is created in SAS® 360 Match if the email address provided by the IDP for the user does not exist in SAS® 360 Match. If the email address does exist, the existing user account in SAS® 360 Match is converted to an SSO account and now must be used in conjunction with the SSO. In other words, a user account that has logged in through the SSO service can no longer log on to SAS® 360 Match in the traditional manner.

As a result, it is important to create at least one user account with administrator access that is NOT used for SSO. Creating a user account with administration access ensures that if the IDP is down, at least one account is still able to log on.

In SAS® 360 Match, SSO is configured in the **SSO** panel on the **Admin** tab.

Here are the fields in the **SSO** panel:

- **Name:** The unique name for the SAS® 360 Match installation. The name must be URL safe because the SSO name is part of the "SP single sign on URL" that is provided to the IDP. This field is read-only after you create the SSO.
- **Active:** Select to indicate that users can authenticated with the SSO.
- **Default:** Select to indicate that anonymous users are authenticated with this SSO by default instead of another SSO or the local application logon.
- **Default role:** Specify the default role for new SAS® 360 Match users signing in from the SSO. Groups, roles, and permissions are not declared by the SSO and must be managed within SAS® 360 Match.

- **Refresh duration:** Specify the amount of time that SAS® 360 Match waits before re-verifying the authentication information with the SSO. The minimum duration is 15 minutes.
- **Idp sso target url:** This information is provided by the IDP.
- **Idp issuer:** This information is provided by the IDP
- **IDP Certificate:** This information is provided by the IDP.
- **SP Certificate:** This information is generated by SAS® 360 Match and can be provided to the IDP to enable assertion encryption.
- **Metadata:** (Optional) The URL this is used to upload IDP information, if the SSO record was created by that method.

Click **Update** after making desired changes.

After the IDP and the SP are configured and enabled, authentication by the SSO starts automatically.

## Data Activation

SAS® 360 Match provides many ways to work with visitor data via cookies or uploaded information.

## Disabling Cookies

During the first visit to a site, each visitor is given a unique ID. This information is maintained across subsequent visits through the `MID` cookie for user ID.

For example, when the visitor goes to a website, SAS® 360 Match creates and stores a cookie on the visitor's browser. The cookie contains the visitor's ID. You can override the cookie and pass a specific ID in the path information of an ad call. If no `MID` is specified in the path information, the ID in the cookie is used.

For customers who want to disable cookies on their sites, SAS® 360 Match provides the ability to pass `MID` information in the path information. This allows the customer to control the ID exchange between the visitor and SAS® 360 Match. In a scenario without cookies, the customer must pass the ID with the ad call if they want to use any features that require an ID or a cookie. To do so, the customer generates the `MID` value and adds it in the form of `/MID=<value>/` to the path information. Also, if it is configured, a state vector ID can be used in place of a `MID` value to identify the visitor.

Contact SAS Technical Support if you want to disable cookies and still retain visitor ID functionality.

## SETID

SAS® 360 Match can set a user cookie named 'external' that can be used to store keys and values. The key needs to be created as a tag in the UX with the proper values. When a `KEY=VALUE` pair is found in the external cookie, which also exists as a tag, the engine automatically uses it as if it is a tag in the path info of the request.

Flights can be targeted using TAG=VALUE pair once the tag and tag values are created to match the name and values in the external cookie. A simple URL request is needed to set keys and values in the external cookie for the visitor. The structure of the URL is:

```
http://adcallURL/{shortname}/setid/external=RED/value=1
```

In this example, RED is the tag name and the value is 1. After this request, the external cookie will have the following values:

```
RED=1  
BLUE=2  
GREEN=3
```

## State Vector

The state vector is a persistent set of data associated with a visitor and can be used for dynamic, real-time targeting of content. State vector consists of an ID to identify the visitor and a collection of tags and events. The state vector ID can be based on a defined ID name (for example, `customerid` or `userid`) or can implicitly use the SAS® 360 Match mid (cookie ID), depending on configuration. Once a state vector is active for a visitor, any calls to SAS® 360 Match automatically consider the tags and events associated with that visitor in the ad selection logic.

### State Vector Tags

Tags are set for a visitor either through an engine directive (`SETSV`) or via a batch process that loads from a flat file. You can use the `SETSV` directive to set a tag's value or values for a visitor. You can also use `SETSV` to append additional values to an existing set of values for the tag. A TTL can be expressed to cause the set values to expire after a specified number of seconds, even during an active session.

You can set tag values that are used only through a current session. When `TTL=SESSION` is present in the `SETSV` call, tag values set by that call are used only within the current visitor session. When the visitor's next session begins, these values are not present.

For example, a visitor could be tagged with `segment=homeowner, sportsfan` after visiting a page with a `SETSV` call on it that could be set to expire after 3 days. If a flight were targeted to either of those "segment" values, this visitor would qualify to receive the flight.

### Authenticating SETSV Calls

By default, users can perform any action using `SETSV` calls. However, you can require that a user provides an API key to process `SETSV` calls. You can also specify a list of allowed tags to enable `SETSV` calls without an API key. Contact SAS Technical Support to enable the API authentication feature or to configure your list of allowed tags.

Include the API key in the path information for the `SETSV` call.

```
/setsv/apiskey=<API KEY>
```

A `SETSV` call that includes any tags that are not on your list of allowed tags must contain an API key to be processed. If the path information contains a mix of tags that are and are not in the allowed list, the tags are processed only if the user includes an API key.

If a `SETSV` call fails, subsequent calls, even if they contain a valid API key, continue to fail until the end of a time-out period, which is 60 seconds by default. Contact SAS Technical Support to change the time-out period.

A call is considered unauthorized if any of the following conditions are met:

- The call requires an API key, but none is given in the path portion of the call.
- The call requires an API key, but the keyword APIKEY is misspelled.
- The call requires an API key, but the value of the API key is not valid.
- Less than one minute has passed since the most recent unauthorized attempt to use SETSV.

This authentication method also applies to GETSV.

## State Vector IDs

By default, the state vector uses the visitor's MID cookie value as their state vector visitor ID. Alternatively, the state vector can use a defined ID name instead of the MID cookie to identify visitors and store a collection of tags and events. With this configuration, the MID cookie is still used to identify and store information about visitors who are not associated with a state vector ID.

You can also configure the MID cookie value to be set to the state vector ID, when it is provided. In this scenario, a MID cookie value is generated for unidentified visitors. Once a state vector ID is provided for the visitor, the MID cookie is set to the same value. This enables you to identify visitors in future sessions without having to supply a state vector ID for every visit.

The state vector ID name and associated MID cookie behavior are configured by SAS Technical Support. Once they are configured, values for the state vector name can be passed in with SETSV calls or through a batch process.

## Secondary State Vector IDs

You can configure secondary state vector IDs to provide additional identity information about a user. Contact SAS Technical Support to enable this feature.

When an ad call includes a secondary ID but not the primary ID, and the data lookup for the secondary ID does not produce a value for the primary ID, then the visitor's state vector is defined by the value of the secondary ID. All state vector data is stored using the ID from the secondary ID.

When an ad call includes a secondary ID and the data lookup for that ID produces a value for the primary ID, the value of the primary ID is used to store state vector data. Also, any state vector data that had previously been stored under the value for the secondary ID is merged into the primary ID's state vector.

However, any ID data that had been defined with the secondary ID value by the user registration process is retained with the secondary ID. Only data that is set in the secondary state vector ID using the SETSV API call is merged into the primary ID state vector. In this way, the secondary ID retains its user registration behavior, even when ad calls have an explicit primary ID value.

Use the user registration data import process to enable a secondary ID value to identify the primary ID value. To do this, you must first configure the mapping between the primary ID value and the secondary ID value.

This feature allows a client who knows only a secondary ID value to make ad calls and still engage with all the data for the visitor through the primary ID. Different calling clients can use

and contribute to a centralized data store for the visitor without themselves knowing the primary identifier of the visitor.

For example, the following two files provide visitor data. The primary state vector ID (SVID in this case) contains the following information:

```
SVID, TAG1
1234, a1234
1235, a1235
1236, a1236
```

The secondary state vector ID (TPID in this case) contains the following information:

```
TPID, TAG2, SVID
4567, b4567,
4568, b4568, 1235
4569, b4569, 1236
```

This table shows the data that is activated by different ad calls and the state vector ID that is used by the session associated with each one:

Ad Call Path Information	Activated Data	State Vector ID	Comment
.../SVID=1234/...	/TAG1=a1234	SVID:1234	No TPID is involved, so normal SVID behavior is exhibited.
.../TPID=4567/...	/TAG2=b4567	TPID:SV:4567	This TPID provided no mapping to a SVID value in the data file, so it becomes its own state vector identification (as indicated by the ":SV" suffix on the name).
.../TPID=4568/...	/TAG1=a1235/TAG2=b4568	SVID:1235	This TPID provided a mapping to an SVID. Therefore, data from the TPID and SVID is activated, and the SVID is used for state vector identification.

For consistency, the same ID should be included explicitly in all requests that are sent during a session. Otherwise, multiple sessions might be created for the same state vector. For example, if some requests are made with ".../TPID=4568/..." and others are made with ".../SVID=1235/..." within the same period, two different sessions could result. The exact behavior depends on customer settings and whether the calling device or browser accepts cookies.

However, using different IDs is acceptable when requests vary across more time, such as when a user is using different devices or browsers at different times. For example, if a user is on a device that sends ".../TPID=4568/..." in the morning and uses a device that sends ".../SVID=1235/..." in the afternoon, the data in both cases is collected under the SVID:1235 identification.

### State Vector Events

Within **Events**, an event name is associated with one or more timestamps corresponding to a visitor action (that is, visiting a certain page). Like tags, they can be recorded either through use of the `SETSV` directive or via a batch process. When set via the `SETSV` directive, the event is recorded as occurring "now". A variation allows the exact time or times to be specified (for example, to express one or more historical events) although the batch process is more

appropriate for that purpose. Events are configured by clicking the **Events** section in the **Targeting** tab of the software or through the XML API.

## Setting Tags via SETSV

To set tags, the SETSV parameters consist of one or more tags and values expressed in the conventional path info format.

### Using the Default MID Method for Visitor IDs

```
https://domain.aimatch.com/customername/SETSV/tag1=value/tag2=value1,value2,...
```

For example:

```
https://domain.aimatch.com/customername/setsv/segment=homeowner,sportsfan/  
gender=m
```

### Using a Defined State Vector ID Name

```
https://domain.aimatch.com/customername/SETSV/svidname=id/tag1=value/tag2=value1,  
value2,...
```

For example:

```
https://domain.aimatch.com/customername/setsv/customerid=1234/segment=homeowner,s  
portsfan/gender=m
```

Tags that are set via this command overwrite the existing values, if any, in the SV. Tags in the SV that are not expressed in the command remain. For example: If the visitor's SV logically already contains:

```
/segment=boatowner/senior=y
```

Then the above example updates the SV to contain:

```
/segment=homeowner,sportsfan/senior=y/gender=m
```

An optional APPEND keyword causes existing tag values to be appended, rather than overwritten. For example: If the SETSV parameters are:

```
.../SETSV/segment=homeowner,sportsfan/gender=m/append
```

Then it updates the original SV to contain:

```
/segment=boatowner,homeowner,sportsfan/senior=y/gender=m
```

The only difference in this example is that the resulting "segment" tag has three values instead of two.

An optional TTL parameter sets a time-to-live, in seconds, for the tag values set by the command. Without a TTL parameter, the values remain set indefinitely. For example: If the SETSV parameters are:

```
.../SETSV/segment=homeowner,sportsfan/gender=m/append/ttl=86400
```

Then it updates the original SV to contain the following:

```
/segment=boatowner,homeowner,sportsfan/senior=y/gender=m
```

But after one day (86440 seconds), it reverts to the following:

```
/segment=boatowner/senior=y
```

If different `APPEND` or `TTL` options are needed for different tags being set, separate `SETSV` commands must be issued.

### *Adding or Removing Tag Values and Events*

In a `SETSV` call, the path info can include `"+="` and `"-="` assignments for tags to add or remove values to a multivalued or stacked tag. A `/TAG+=VALUE1` appends `VALUE1` to any existing tag values for `TAG`. If `TAG` is multivalued and already has a `VALUE1`, another is not added. If `TAG` is a stacked tag, `VALUE1` is added to the end of the tag's stacked values, and repeat values are possible.

`/TAG-=VALUE1` removes `VALUE1` from any existing tag values for `TAG`. If `TAG` does not have a `VALUE1`, its values are not changed. If `TAG` is a stacked tag, all occurrences of `VALUE1` are removed.

The right-hand side can express multiple values, in which case those values are all added to or removed from the tag.

Use the `"+="` assignment to append events and the `"-="` assignment to remove events with the `EVENT.eventname` tag. For example:

```
.../EVENT.CONVERTED+=1534537253,1534537399/...
```

adds two new event times to the `CONVERTED` event, subject to the event's maximum count and time policy.

```
.../EVENT.CONVERTED-=1534537253,1534537399/...
```

removes two event times from the `CONVERTED` event, assuming they were there in the first place. Timestamps that are not in the visitor's event occurrence list are ignored.

### **Recording Events via SETSV**

To record a new event, the `SETSV` parameters consist of an `EVENT` tag whose value is the name of the event:

```
https://domain.aimatch.com/customername/SETSV/event=eventname
```

For example:

```
https://domain.aimatch.com/customername/SETSV/event=abandonedcart
```

Multiple events can be recorded in the same call.

For example:

```
.../SETSV/event=abandonedcart,viewedcart
```

The events are recorded as having occurred at the time the command was received.

An alternative syntax allows one or more explicit timestamps to be specified for each event. The `EVENT` tag must be suffixed with the event name to the left of the equal symbol:

```
.../SETSV/event.eventname=timestamp,timestamp,...
```

Each timestamp must be expressed as a UNIX epoch timestamp—the number of seconds that have elapsed since January 1, 1970 (midnight GMT).

When explicit timestamps are specified, the `APPEND` keyword should be present to add them to previous event occurrences. Without the `APPEND` keyword, the explicit list of timestamps replaces all existing occurrences of the event. `APPEND` is assumed when an event is expressed without explicit timestamps:

```
.../SETSV/event.abandonedcart=1440278686,1440341491/append
```

adds two occurrences of the event `abandonedcart` to any existing set of occurrences. The times are for Sat, 22 Aug 2015 21:24:46 GMT and Sun, 23 Aug 2015 14:51:31 GMT.

Without the `APPEND` option, this example replaces existing occurrences of the `abandonedcart` event with just these two. Similarly, all existing occurrences of `abandonedcart` can be removed using:

```
.../SETSV/event.abandonedcart=
```

## Explicitly Merging State Vectors via SETSV

When a configured state vector ID name is being used for the state vector (and not the `MID` value), `SETSV` can be used to explicitly merge two state vectors:

```
.../SETSV/svidname=id1/mergefrom=id2
```

where `svidname` is the configured SV ID name, `id1` is the ID of the target state vector (the one to be merged into), and `id2` is the ID of the source state vector (the one to be merged from).

Tags from `id2` are assumed to be newer than those from `id1` and overwrite the same tags in `id1`. For events, `id1` is updated to have the union of all events from `id1` and `id2`. The merge operation not only updates `id1`'s state vector as described, but also clears out the state vector for `id2`. For example, if `customerid` is the configured SV ID, and the SV for ID 12345 is (logically):

```
/segment=homeowner,sportsfan/gender=m/event.abandonedcart=1440278686
```

And for ID 98765 is:

```
/segment=boatowner/senior=y/event.abandonedcart=1440341491/  
event.viewedcart=1440341491
```

Then the `SETSV` command:

```
.../SETSV/customerid=12345/mergefrom=98765
```

results in the following updated SV for 12345:

```
/segment= boatowner/senior=y/gender=m/  
event.abandonedcart=1440341491,1440278686/event.viewedcart=1440341491
```

When one state vector with no tag values is merged with another state vector with tag values, the merged state vector contains tag values from the state vector with values. However, there might be conflicts if both state vectors have values.

Contact SAS Technical Support to specify one of the following behaviors if a merge conflict occurs when you combine tag values from two state vectors:

- Merge and retain the values from both state vectors. This is the default behavior.
- Keep the current values and ignore the values from the MERGEFROM state vector.
- Ignore the current values and keep the values from the MERGEFROM state vector.

In addition, contact SAS Technical Support to define a list of predefined tag values to merge with the tag values from the state vectors. This list is empty by default and is ignored when you use the default behavior of merging and retaining values from both state vectors.

## Implicitly Merging State Vectors via SETSV

When a visitor's state vector accumulates data under its `MID` value, the expression of a state vector ID causes the `MID`-based data to be implicitly merged into the SV ID-based data. Tags in the `MID`-based data are regarded as newer, overwriting any existing values in the SV ID-based data. Events are merged as described above.

## Loading Multiple Data Sets

The SET directive enables you to update the data for third-party IDs and persisted supertags. The SET call includes a third-party ID or supertag ID along with a payload of tags to update. SAS® 360 Match updates the existing data for the ID and updates it with the other tags in the SET call (similar to SETSV).

SAS® 360 Match checks the third-party IDs and then the supertag IDs in the order that the IDs are configured. Match updates the data for the first ID that it finds with the tags from the SET call. It applies a TTL value if one is provided.

If a SET call references a state vector ID (primary or secondary), the call is treated as a SETSV call. Therefore, SET or SETSV can be used to set state vector data when an explicit state vector ID is used.

The GET directive works in a similar manner, where the data from the first ID that is found is returned. The call is treated as a GETSV call when it references a state vector ID.

## Bulk Data Upload via SV

Uploading bulk SV data into SAS® 360 Match allows customers to specify a set of targeting tags to use with the current visitor based on a specified ID. This feature is commonly used to import user registration data. It can also be used to import a set of tag values based on any type of ID, even an ID that is not necessarily visitor-specific, such as a program ID. IDs are not case sensitive.

## Data File Requirements

You can use the bulk upload process to update SV and non-SV data. SV data includes tags and events, and requires an SV ID name (for example, `customerid`). Non-SV data can include only tags and also requires an ID. Contact SAS Technical Support to define the IDs.

The uploaded file uses the CSV format, with the first line consisting of headers that define all the columns. Any double quotation marks around the column names are ignored.

For SV uploads, the table must include one column for the state vector ID, which must be the first column that references an ID. The best practice is to make this the first column in the table. The column names are otherwise assumed to be tags or events.

No `APPEND` option is assumed for tags or events. Column names in uploaded files can end with a “+” or “-” to indicate whether values in that column should be added to or removed from the corresponding tag, respectively. Without one of these suffixes, the data appearing for a tag or event are considered to represent all the values to be stored, replacing any prior values.

For tag values, the `TTL` is configurable, but defaults to 6 months.

You can configure the upload so that changes made to the SV through uploaded files become effective immediately for visitors with an active session. Files are processed in the order in which they are received.

SAS® 360 Match supports two CSV formats for imported files, indicated by adding the `fmt1` and `fmt2` format codes to the file extension. Each format parses the uploaded data in a different way.

The default format, noted by `fmt1`, supports multivalues and stacked values. This format uses commas to separate tag columns. You can stack values in tag data using the “!” symbol. Multivalues are separated by a URL-encoded comma (`%2C`). Alternatively, the entire column’s data can be enclosed in double quotation marks, and commas can be used within the data to delimit multiple values. Event columns must be identified as `EVENT.eventname`. The values for events must be timestamps in the same format as `SETSV`.

The second format, noted by `fmt2`, does not support multivalues, stacked values, or values with embedded line breaks. Like the default format, this format uses commas to separate columns. However, tag data is considered a single value. If the value includes a comma or double quotation marks, the entire column’s data must be enclosed in double quotation marks. If the data includes double quotation marks, it must be escaped with an additional double quotation mark.

At the file level, the format code has to precede the `.CSV` or `.CSV.gz` suffix, such as `filename.<formatcode>.csv`. When no format code is present in the file name, the default format is used.

At the column level, the format code can be added as a suffix to the column name, such as `columnname:<formatcode>`. When no format code suffix is present, the processing format to use is determined from the file name. The file name establishes a format for the file, and individual columns can override it.

A file named `data.csv` with the header row `id,tag1,tag2,tag3` parses all columns using the default format.

A file named `data.fmt2.csv` with the header row `id,tag1,tag2,tag3` parses all columns using the single-value format.

A file named `data.csv` with the header row `id,tag1,tag2:fmt1,tag3:fmt2` parses the `id`, `tag1`, and `tag2` columns using the default format, and `tag3` using the single-value format.

A file named `data.fmt2.csv` with the header row `id,tag1,tag2:fmt1,tag3:fmt2` parses the `id`, `tag1`, and `tag3` columns using the single-value format, and `tag2` using the default format.

## User Registration

User registration enables customers of SAS® 360 Match to target specific visitors based on accumulated demographic data such as age or gender. This feature is used to indirectly specify a set of targeting tags to use with the current visitor based on a specified ID. Typically, the user

registration feature is used to import user registration data. However, the feature can also be used to import a set of tag values based on any type of ID, even for an ID that is not visitor specific such as a program ID.

To enable the user registration feature, the following items are required:

- The customer must create the data file that contains unique identifiers for visitors and their associated demographic data.
- For configuration purposes, provide SAS Technical Support with the string for the visitor's unique identifier such as `/userid={some unique value}/` that is passed in on the ad request URL path information. If this `key=value` pair is present in the request, the value is used as the ID of the visitor for user registration demographic data lookup.
- The customer must create the custom tags and values that correspond to the demographic data column names in the user registration file.
- The customer must build the ad request URLs so that the unique visitor identifier `key=value` pair is passed to SAS® 360 Match in the URLs.

By default, items in the file expire six months after they are last updated. You can use an optional column named `TTL` to specify the lifetime, in seconds, of individual items in the file. For example, if the TTL value is set to 600, the item is deleted after 10 minutes. If the TTL value is not set, is less than 0, or is not a numeric value, the default lifetime of six months after the last update is used.

The user registration data file format is contained in a text file. The file is comma delimited, with one record per line. The first line is a header line that contains the names of the tags used for user data. The names of the header values must match the names of the tags in the SAS® 360 Match database. The values in the record lines must match if logging is to work. Otherwise, the values can be used for targeting purposes with a `LIKE` operator.

Here is an example of a file:

```
userid,age,gender
1,50,m
2,80,f
3,200,o
```

The first line of the file is a header row that defines the unique user IDs and the keys or targeting values. The first column must be the unique visitor identifier. The demographic values are contained in each row.

In this example, if the path information of the ad request URL includes `userid=1`, SAS® 360 Match performs a lookup on this value on the initial request and returns `age=50` and `gender=m` for this specific visitor for targeting. You can combine this information with corresponding custom tags in the UX to create targets that use this data. In this example, `age` and `gender` are the custom tags.

Multiple trait values can be logged if the values are encased in quotation marks and separated by commas. For example:

```
ID,hobby,job_title
1,"hockey,music",PM
```

User registration data is stored under the value that is specified by the ID column in the text file. The values are case-insensitive. When comparing values, SAS® 360 Match converts characters to uppercase with the assumption that it is converting plain ASCII characters. This conversion

might not occur or might occur incorrectly for non-ASCII characters. Therefore, SAS® 360 Match might not be able to make proper comparisons for non-ASCII values.

For example:

```
ID,GENDER,REGION,FAVORITE
123,Male,NorthEast,Thriller
85tyk9,Female,Caribbean,Romance
```

If the user registration data file is not properly formatted, the loading process fails and no user registration data is available for targeting.

## Uploading Data Files

Data files should be uploaded through the REST API. However, SAS Technical Support can create a location for the files to be uploaded using Amazon S3. In this case, the file is moved to an archive directory after processing so that its absence from the upload location indicates processing is complete. The uploaded files can end with CSV or CSV.gz. The files must have one of those extensions. No other files are processed. If multiple files are present, they are all processed. File name conventions should follow Linux naming conventions. For more information, see [http://www.lininfo.org/file\\_name.html](http://www.lininfo.org/file_name.html). There is no limit to the number of rows in the files. If a new file has the same name as an existing file that has not yet been processed, and therefore not yet moved to the archive bucket, the new file overwrites the existing file.

When importing the data, no characters are ignored. Whatever is found for a tag value is kept as is and presented to the engine. If the input contains characters that would ordinarily be disallowed when creating tag values, then it is, by definition, an unrecognized value and the engine does not match it to any targets. If duplicate user ID rows are present, SAS® 360 Match overrides the first entry's values with the latter entry's values.

When processing the uploaded file, the only row-level issue that can occur is a failure or time-out when attempting to insert new data for a visitor. In such a case, the new data is not inserted and the process continues with the next row. The import process fails if the user ID value is not found in the file header or if the field delimiter is not a comma. Once processed, the file is archived. Any archived user data files are purged after 30 days.

If you want to null out a user's data, you can upload the CSV with the user ID, and list segments (keys) with no data in the value field associated with them. If you are trying to add a new segment, you can upload the standard CSV with only that data in it (for example, user ID, key, value). For example:

### Case 1

You would like to delete three users who no longer want to be profiled for advertising:

```
userid, matchtype, barb, de, th
qw41, email, a3.aa.ma.me, 56595, 6775
we25, nonmatch, a3.w3.wo, 94489, 6647
er47, nameemail, aa.hc.wa.wo, 76692, 6664
```

Submit the CSV as follows:

```
userid, matchtype, barb, de, th
qw41, , , ,
we25, , , ,
er47, , , ,
```

This eliminates all the fields associated with the users except their user ID.

## Case 2

You would like to add new data to a user for additional targeting. Submit the following CSV to update the details:

```
userid, matchtype, barb, de, th, tgt  
qw41, email, a3.aa.ma.me, 56595, 6775, sue
```

Submit this CSV for the same result:

```
userid, tgqwqw41, sue
```

## Case 3

Same as Case 2 except instead of a new field, use an existing field:

```
userid, matchtype, barb, de, th,  
qw41, email, a3.aa.ma.me, sue, 6775
```

Submit this CSV for the same result:

```
userid, de,  
qw41, sue
```

## Duplicate Tags

If the ad request contains tags that match those in the state vector data set, then the tag values on the request become the defaults for that visitor. If a request comes in without those tags, the state vector values are used. For example, if the ad call expresses `GENDER`, then it overrides the value for just that request, and its value is the one logged. If the next call comes in without `GENDER`, the state vector value is used again.

## Tokens

`Key=value` pairs looked up by state vector are available to be tokenized using the standard `%%TAG%%` token format.

## Deleting Personally Identifiable Information

When the value for `DELETEALL` in a `SETSV` call is set to any nonzero integer, the user ID record, user registration data, and all the information that is stored in the specified visitor's state vector is deleted, including tags, events, exposure history, tracking history, and persistent creatives. Other tags in the same call are ignored and do not update the state vector. If `DELETEALL` is not present or if `DELETEALL=0`, normal `SETSV` processing occurs.

All data can be deleted for a given ID by including a column named `DELETEALL` and using a nonzero integer value. If the value for that column is empty or "0", then other columns are processed normally.

These deletion mechanisms do not prevent new data from being set for visitors. For true do-not-track behavior, you must also stop sending data updates for visitors' state vector and in their user registration afterward.

## Identity Mapping

The identity mapping feature complements the user registration and state vector features. This feature allows the upload of a file with user data that references an identity in another user data file. In this manner, it would be possible to identify a user on a site and load centralized data for

that user automatically for use in targeting. For more information about identity integration, contact SAS Technical Support.

## Data Expiration

Uploaded user data is purged after 6 months if the data has not been refreshed or overwritten in that time. This expiration time can be configured by SAS Technical Support.

You can also use the `REFRESHALL` tag to reset the TTL for data that is associated with an active visitor. By default, the value for this tag is 0 and the TTL is not refreshed. Set the tag value to 1 to refresh the visitor's TTL.

Use the `SETSV` directive to refresh data for a visitor:

```
.../SETSV/customerid=12345/refreshall=1/...
```

Provide a `REFRESHALL` value in a bulk file upload to refresh data for multiple visitors:

```
customerid,refreshall
12345,1
12389,1
...
```

## Add UI Extensions

By default, the security framework for SAS® 360 Match prevents you from adding extensions to the user interface. Contact SAS Technical Support if you want to add individual domains to a trusted list and enable your extensions.

## Create a Reverse Proxy

SAS® 360 Match enables you to create a reverse proxy from your website to your ad server so that the request from the ad server appears to be part of your site and is not subject to browser-based restrictions. When you use the proxy mechanism, requests from your ad server are considered as same-origin requests, do not have cookies that expire, and do not limit the amount of script-writable storage.

You can nest the path for the ad server as deeply as you like within your website. For example, you can use a CDN configuration for `www.example.com` that creates a reverse proxy without caching for `/sas` to `https://example-ads.aimatch.com/` so that `www.example.com/sas/hserver` returns an HTML ad as expected.

To enable this feature, decide where to host the ad server within your website and contact SAS Technical Support with this information.



SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © 2026, SAS Institute Inc. All rights reserved.

---