Module 5 - Mobile services and API

This toolkit is designed for Professional Developer Exam Aspirants. There are **six** Modules. Study Each module per week to stick to schedule. Technical Parts of applications are depicted in Videos, you can learn more about them from experience League. You can visit Get prep page to understand the contents and anticipate the learning journey.
This is Professional Exam, Developer toolkit Module 5. This module contains five sections.

5.1 Adobe Analytics: Email Marketing (ESP) Integration Case Study

Adobe Analytics: Email Marketing (ESP) Integration Case Study

Integrating with an Email Service Provider (ESP) has been one of the most popular integration use cases for Adobe Analytics customers for a long time. Many of these integrations were available as pre-built Data Connectors for Adobe Analytics. However Adobe has discontinued its support for Data Connectors and so many of our partners are now looking for an alternative integration method.

In this post, I'll break down what most of the email marketing Data Connector integrations do behind the scenes and describe how you can create something similar on your own when a Data Connector integration isn't available.

The basic idea

In Adobe Analytics you can see how many people have clicked through from an email campaign because you can track the campaign codes included in query string parameters on the link from the email. You can then see what users did from there on your site and tie that behavior back to click throughs. But what about before the user clicks through to your site? There are metrics like emails sent, emails opened, and emails unsubscribed that your email marketing provider has but that Adobe Analytics doesn't have. Those metrics need to be sent from your email marketing provider to Adobe Analytics so that they can also be taken into account when you're looking at your campaign's performance.

Going in the other direction, it can be really useful to pass information about your customers' behavior on your site from Adobe Analytics to your email marketing tool to use for re-marketing. For example, you might want to target people who abandoned their cart without purchasing. You can pass information about that group of people back to your email marketing tool from Adobe Analytics and target those people campaigns.

In the next section, I'll describe how you can get email metrics from your email marketing provider into Adobe Analytics. In the section after that I'll describe how to pass information from Adobe Analytics to your email marketing provider.

Getting data from your email marketing provider into Adobe Analytics

The first (and maybe most important) two pieces of data that you need to make sure you are getting from your email provider are the campaign ID and recipient ID. These IDs are usually included as query string parameters on the link in the email. You can use processing rules to copy these values into two separate eVars in Adobe Analytics without adding any code to your webpages. Alternatively, you could set up a rule in Adobe's tag manager Adobe Experience Platform Launch to capture these values. Whichever method you use, these two values need to be captured and placed into two separate eVars when a user comes to your site from one of your email campaigns.

The recipient ID will be used for re-marketing but you probably won't use it for any reporting within Adobe Analytics so we'll leave it aside for now and come back to it in the next section. For now, we'll focus on the campaign ID. These IDs are not very user-friendly—it would be difficult for your analysts to use them in reporting. So the next step is to get user-friendly data from your email marketing provider and use it to 'classify' (or provide metadata about) these IDs which your analysts can then use for reporting. For this, you'll use classifications. You will need to work with email provider to upload a file that associates those campaign IDs with all sorts of metadata such as Message Name, Message Subject, Delivery Tool, Campaign Name, Category etc. Each of these pieces of metadata (or 'classifications') can then be used in Adobe Analytics as a report making life much easier on your analysts and giving them a lot of flexibility in the way they report on your email campaign performance. Check out this documentation for info on the Classifications API.

The second set of data that you'll want to get from your email provider is all of the metrics about your campaigns—emails sent, delivered, bounced, opened, unsubscribed, and clicked-that Adobe Analytics doesn't have. For these metrics, you'll want to use Data Sources (not to be confused with Data Connectors). Data Sources allow you to pull offline metrics into Adobe Analytics and associate them with eVars. The eVar that you'll want to associate these metrics with is your campaign eVar. Once these metrics are pulled into Adobe Analytics, you'll be able to see how your campaigns are doing from start to finish—from how many emails you sent to how many customers from that campaign converted on your website. Check out this documentation for information on the Data Sources API.

And that's it for getting data into Adobe Analytics. The next section describes how to get data back to your email marketing provider.

Getting data from Adobe Analytics to your email marketing provider

You can use Adobe Analytics to segment your users and then send this segment information back to your email marketing tool for retargeting. For example, you might want to create a segment of users that abandoned their carts without finishing the purchase. You could then use this information to target an email campaign to those users reminding them about their items sitting in their cart and maybe offering a discount or other incentive to finish the purchase.

Adobe Analytics' segmenting tool is very flexible so the types of segments you create to send back to your email marketing provider are limited only by your data and your imagination. Check out the documentation on the segmenting tool for more information.

Once you have those segments created, you'll need to send them back to your email marketing provider. You can do this by creating a Data Warehouse feed. Remember the recipient ID? Here's where you'll use it. You can set up the feed to include that recipient ID eVar and any segments that you want to associate with that ID (e.g., cart abandons, product views, product purchases etc.). Often Adobe Analytics clients also include the products variable, the email campaign ID eVar, and the date in their feeds. You'll need to coordinate with your email provider on the data you'll be sending them and they'll need an ftp site available to receive the data. Usually Adobe Analytics clients opt to send this data once a day but Data Warehouse can do other frequencies if you need. Check out this documentation on the Data Warehouse API (which is a part of the reporting API).

And that's it. Hopefully this gives you a good foundation for building your own custom integration between Adobe Analytics and your email provider. Reach out to Adobe Consulting Services if you need help building this custom integration.

For more information about Adobe Analytics integration tools in general and how to get an Adobe Analytics sandbox for building and testing your integration, check out this article.

5.2 Adobe Analytics APIs

The endpoints described here are routed through analytics.adobe.io. In order to use these endpoints you must create an oAuth client that is subscribed to access the Adobe Analytics Reporting API.

Annotations

GET `/annotations`
Retrieves all annotations
POST `/annotations`
Create new annotation
GET `/annotations/{id}`
Get existing annotation
PUT `/annotations/{id}`
Update existing annotation
DELETE `/annotations/{id}`
Delete existing annotation

Calculated Metrics

GET `/calculatedmetrics`
Retrieve many calculated metrics
POST `/calculatedmetrics`
Create a new Calculated Metric

GET **/calculatedmetrics/functions**
Retrieve calculated metric functions
GET **/calculatedmetrics/functions/{id}**
Retrieve a calculated metric function by id
POST **/calculatedmetrics/validate**
Validate a calculated metric definition
GET **/calculatedmetrics/{id}**
Retrieve a single calculated metric by id
PUT **/calculatedmetrics/{id}**
Update an existing calculated metric
DELETE **/calculatedmetrics/{id}**
Deletion of Calculated Metrics by Id

Component Meta Data

GET **/componentmetadata/shares**
Returns all components that have been shared by the current user (meeting the paging restriction)
POST **/componentmetadata/shares**
Share 1 component with 1 user
PUT **/componentmetadata/shares**
Share components with users. WARNING: Authoritative; deletes/overwrites all pre-existing shares for the given components
POST **/componentmetadata/shares/component/search**
Search for shares for several components at once
GET **/componentmetadata/shares/{id}**
Returns details about a specific share
DELETE **/componentmetadata/shares/{id}**
Deletes the share with the given id
GET **/componentmetadata/shares/sharedto/me**
Returns all component ids that have shared to the current user
POST **/componentmetadata/tags/component/search**
search for tags for several components at once
GET **/componentmetadata/tags**
Returns a list of tags for the current user's company
POST **/componentmetadata/tags**
Saves the given tag(s) for the current user's company
DELETE **/componentmetadata/tags**
Disassociates all tags from the given components
GET **/componentmetadata/tags/search**
Retrieves a tags for a given component by componentId and componentType
PUT **/componentmetadata/tags/tagitems**
Tag a component with one or many tags at once. WARNING: Authoritative; deletes/overwrites all pre-existing associations
GET **/componentmetadata/tags/{id}**
Retrieves an tag by its id
DELETE **/componentmetadata/tags/{id}**
Removes the tagId and all associations from that tag to any components
GET **/componentmetadata/tags/tagnames**
Retrieves component ids associated with the given tag names

Date Ranges

GET **/dateranges/{id}**
Retrieves configuration for a DateRange.

PUT **/dateranges/{id}**
Updates configuration for a date range.
DELETE **/dateranges/{id}**
deletes a date range.
GET **/dateranges**
Returns a list of dateranges for the user
POST **/dateranges**
Creates a single date range.

Dimensions

GET **/dimensions**
Returns a list of dimensions for a given report suite.
GET **/dimensions/{id}**
Returns a dimension for the given report suite

Metrics

GET **/metrics**
Returns a list of metrics for the given report suite
GET **/metrics/{id}**
Returns a metric for the given report suite

Projects

GET **/projects/{id}**
Retrieves configuration for a Project.
PUT **/projects/{id}**
Updates configuration for a project.
DELETE **/projects/{id}**
deletes a project.
POST **/projects/validate**
Validates a Project definition
GET **/projects**
Returns a list of projects for the user
POST **/projects**
Creates a single project.

Reports

POST **/reports**
Runs a report for the request in the post body
GET **/reports/topItems**
Runs a top items report for the request in the post body

Segments

GET **/segments**
Retrieve All Segments
POST **/segments**
Creates Segment
POST **/segments/validate**
Validate a Segment
GET **/segments/{id}**
Get a Single Segment
PUT **/segments/{id}**
Update a Segment

DELETE/segments/{id}
Delete Segment

Users

GET/users
Returns a list of users for the current user's login company

GET/users/me
Get the current user

Usage Logs

GET/auditlogs/usage
Retrieves usage and access logs for the search criteria provided.

Models

AlternateVariableNames{

name	string
baseName	string
curatedName	string

}

AnalyticsAnnotation{

id	string
name	string
description	string
dateRange	string
color	stringEnum: Array [9]
applyToAllReports	boolean
scope	**AnnotationScope**{...}
createdDate	string($date-time)
modifiedDate	string($date-time)
modifiedById	string
tags	[...]
shares	[...]
approved	boolean
favorite	boolean
usageSummary	{...}
systemUserOwned	boolean
owner	**Owner**{...}
companyId	integer($int32)
reportSuiteName	string
rsid	string

}

AnnotationScope{

metrics	[...]
filters	[...]

}

AnalyticsAsrErrorResponse{

errorDescription	string
errorCode	string
errorId	string

}

AnalyticsCalculatedMetric{

id string
readOnly: true

System generated id

name string
description string
rsid string

The report suite id for which the component was created/updated

reportSuiteName string
readOnly: true

The report suite name for which the component was created/updated

owner **Owner**{...}
polarity string

Set metric polarity, which indicates whether it's good or bad if a given metric goes up. Default=positive

Enum:
Array [2]
precision integer($int32)

Number of decimal places to include in calculated metric result

type stringEnum:
Array [4]
definition* **CalculatedMetricDef**{...}
categories [...]
tags [...]
siteTitle string
modified string($date-time)
created string($date-time)
readOnly: true

Calculated metric creation date

}

AnalyticsDimension{

id string
title string
name string
type stringEnum:
Array [8]
category string
categories [...]
support [...]
pathable boolean
parent string
extraTitleInfo string
segmentable boolean
reportable [...]
description string

allowedForReporting boolean
noneSettings NoneSettings{...}
tags [...]
}

AnalyticsMetric{

id string
title string
name string
type stringEnum:
Array [8]
extraTitleInfo string
category string
categories [...]
support [...]
allocation boolean
precision integer($int32)
calculated boolean
segmentable boolean
description string
polarity stringEnum:
Array [2]
helpLink string
allowedForReporting boolean
tags [...]
}

AnalyticsProject{

id string
readOnly: true

System generated id

name string
description string
rsid string

The report suite id for which the component was created/updated

reportSuiteName string
readOnly: true

The report suite name for which the component was created/updated

migratedIds [...]
companyTemplate boolean
template boolean
type stringEnum:
Array [2]
definition JsonNode{...}
externalReferences {...}
accessLevel string
versionNotes string
tags [...]
shares [...]
approved boolean
favorite boolean
usageSummary {...}
complexity {...}
owner Owner{...}
siteTitle string

modified string($date-time)
created string($date-time)

}

AnalyticsSegment{

name string

A name for the segment.

description string

A description of the segment.

rsid string

The report suite id.

reportSuiteName string

The friendly name for the report suite id.

owner Owner{...}
definition AnalyticsSegmentDefinition{...}
compatibility SegmentCompatibility{...}
definitionLastModified string($date-time)
categories [...]
siteTitle string

A name for the report suite. This is deprecated and should use the report suite name instead.

tags [...]
modified string($date-time)
created string($date-time)

}

AnalyticsSegmentResponseItem{

id string

Id of the segment.

name string

A name for the segment.

description string

A description of the segment.

rsid string

The report suite id.

reportSuiteName string

The friendly name for the report suite id.

owner Owner{...}
definition AnalyticsSegmentDefinition{...}
compatibility SegmentCompatibility{...}
definitionLastModified string($date-time)
categories [...]
siteTitle string

A name for the report suite. This is deprecated and should use the report suite name instead.

tags [...]
modified string($date-time)
created string($date-time)

}

AnalyticsSegmentDefinition{

container {...}
func string
version [...]

}

AnalyticsUser{

companyid integer($int32)
loginId integer($int32)
login string
changePassword boolean
createDate string($date-time)
disabled boolean
email string
firstName string
fullName string
imsUserId string
lastName string
lastAccess string($date-time)
phoneNumber string
tempLoginEnd string($date-time)
title string

}

CalcMetricFunction{

id string

Calculated Metric Function ID

category string

Calculated Metric Function category

persistable boolean

If a Calculated Metric Function is persistable

name string

Calculated Metric Function name

namespace string

Calculated Metric Function namespace

description string

Calculated Metric Function description

exampleKey string

Calculated Metric Function example key

example string

Calculated Metric Function example

definition CalcMetricFunctionDef{...}

}

CalcMetricFunctionDef{

func string
parameters [...]
formula {...}
version [...]

}

CalcMetricFunctionParameter{

func string
name string
type string
friendlyNameKey string
descKey string
friendlyName string
description string
default-value {...}

}

CalculatedMetricDef{

}

CalculatedMetricErrorStatus{

errorCode stringEnum:
Array [29]
errorDescription string
errorId string
errorDetails {...}
rootCauseService string

}

CalendarType{

rsid string
type stringEnum:
Array [6]
anchorDate string($date-time)

}

ComponentSearch{

componentType string
componentIds [...]

}

CuratedComponent{

componentId* string
componentType string
curatedName string

}

DeleteResponse{

```
    result              string
    message             string
}
```

ExpandedDateRange{

```
    id                  string
    name                string
    description         string
    modified            string($date-time)
    companyId           integer($int32)
    owner               {...}
    definition          {...}
    template            boolean
    createDate          string($date-time)
    disabledDate        string($date-time)
    alternateVariableNames AlternateVariableNames{...}
    curatedItem         boolean
    imsOrgId            string
    systemUserOwned     boolean
    tags                [...]
    shares              [...]
    approved            boolean
    favorite            boolean
    usageSummary        {...}
}
```

JsonNode{

```
    valueNode           boolean
    floatingPointNumber boolean
    containerNode       boolean
    missingNode         boolean
    pojo                boolean
    integralNumber      boolean
    short               boolean
    int                 boolean
    long                boolean
    double              boolean
    bigDecimal          boolean
    bigInteger          boolean
    textual             boolean
    binary              boolean
    float               boolean
    nodeType            stringEnum:
                        Array [ 9 ]
    boolean             boolean
    number              boolean
    object              boolean
    array               boolean
    null                boolean
}
```

Locale{

```
    language            string
    script              string
    country             string
    variant             string
    extensionKeys       [...]
    unicodeLocaleAttributes [...]
    unicodeLocaleKeys   [...]
    iso3Language        string
    iso3Country         string
    displayLanguage     string
    displayScript       string
    displayCountry      string
    displayVariant      string
```

displayName string

}

NoneSettings{

includeNoneByDefault boolean

noneChangeable boolean

}

Owner{

id* integer($int32)

the login id of the owner

name string

the friendly full login name of the owner, included when the expansion parameter ownerFullName is true

login string

the friendly full login name of the owner, included when the expansion parameter ownerFullName is true

}

ProjectCompatibility{

valid boolean

validatorVersion string

message string

}

RankedColumnError{

columnId string

errorCode stringEnum:
Array [8]

errorId string

errorDescription string

}

RankedColumnMetaData{

dimension ReportDimension{...}

columnIds [...]

columnErrors [...]

}

RankedReportData{

totalPages integer($int32)

firstPage boolean

lastPage boolean

numberOfElements integer($int32)

number integer($int32)

totalElements integer($int32)

message string

request RankedRequest{...}

reportId string

columns RankedColumnMetaData{...}

rows [...]

summaryData RankedSummaryData{...}

}

RankedRequest{

rsid string

dimension string

locale Locale{...}

```
    globalFilters       [...]
    search              ReportSearch{...}
    settings            RankedSettings{...}
    statistics          RankedStatistics{...}
    metricContainer     ReportMetrics{...}
    rowContainer        ReportRows{...}
    anchorDate          string
}
RankedSettings{
    limit                   integer($int32)
    page                    integer($int32)
    dimensionSort           string
    countRepeatInstances    boolean
    reflectRequest          boolean
    includeAnomalyDetection boolean
    includePercentChange    boolean
    includeLatLong          boolean
    nonesBehavior           string
}
RankedStatistics{
    functions           [...]
    ignoreZeroes        boolean
}
RankedSummaryData{
}
ReportDimension{
    id                  string
    type                stringEnum:
                        Array [ 8 ]
}
ReportErrorStatus{
    errorCode           stringEnum:
                        Array [ 29 ]
    errorDescription    string
    errorId             string
    errorDetails        {...}
    rootCauseService    string
}
ReportFilter{
    id                  string
    type                stringEnum:
                        Array [ 4 ]
    dimension           string
    itemId              string
    itemIds             [...]
    segmentId           string
    segmentDefinition   AnalyticsSegmentDefinition{...}
    dateRange           string
    excludeItemIds      [...]
}
ReportMetric{
    id                  string
    columnId            string
    filters             [...]
    sort                string
    metricDefinition    {...}
    predictive          ReportMetricPredictiveSettings{...}
}
```

ReportMetricPredictiveSettings{

anomalyConfidence	number($double)

}

ReportMetrics{

metricFilters	[...]
metrics	[...]

}

ReportRow{

rowId	string
filters	[...]

}

ReportRows{

rowFilters	[...]
rows	[...]

}

ReportSearch{

clause	string
excludeItemIds	[...]
itemIds	[...]
includeSearchTotal	boolean
empty	boolean

}

Row{

itemId	string
value	string
rowId	string
data	[...]
dataExpected	[...]
dataUpperBound	[...]
dataLowerBound	[...]
dataAnomalyDetected	[...]
percentChange	[...]
latitude	number($double)
longitude	number($double)

}

RowItem{

itemId	string
value	string

}

SavedSharedComponent{

componentType	string
componentId	string
shares	[...]

}

ScopeFilter{

id	string
operator	string
dimensionType	string
terms	[...]
componentType	string

}

ScopeMetric{

id	string
componentType	string

}

SegmentCompatibility{

valid	boolean
message	string
validator_version	string
supported_products	[...]

supported_schema [...]
supported_features [...]
}

Share{

shareId string
imsOrgId string
shareToId integer($int32)
shareToImsId string
shareToType string
shareFromImsId string
componentType string
componentId string
shareToDisplayName string
shareToLogin string
accessLevel string
}

SharedComponent{

componentType string
componentId string
shares [...]
}

SharedToEntity{

shareToId integer($int32)
shareToImsId string
shareToType string
accessLevel string
}

Tag{

description:
Tag Model

id integer($int32)

the tag id

name string

the tag name

description string

the tag
description

components [...]
}

TaggedComponent{

componentType string
componentId string
tags [...]
}

TimezoneSettings{

currentTimezoneOffset number($float)
name string
timezoneZoneinfo string
timezoneId integer($int32)
}

UnhashReportData{

totalPages integer($int32)

```
    firstPage           boolean
    lastPage            boolean
    numberOfElements    integer($int32)
    number              integer($int32)
    totalElements       integer($int32)
    message             string
    reportId            string
    searchAnd           string
    searchOr            string
    searchNot           string
    searchPhrase        string
    oberonRequestXML    string
    oberonResponseXML   string
    rows                [...]
}
```

ResponsePageUsageLogDto {

```
    content             [...]
    totalElements       integer($int64)
    lastPage            boolean
    numberOfElements    integer($int64)
    totalPages          integer($int32)
    firstPage           boolean
    sort                string
    size                integer($int32)
    number              integer($int32)
}
```

UsageLogDto {

```
    dateCreated         string($date-time)
    eventDescription    string
    ipAddress           string
    rsid                string
    eventType           string
    login               string
}
```

5.3 Welcome to the enterprise and teams admin guide

Visit here to watch a video and learn how you can use the Adobe Admin Console to manage your Adobe entitlements across your entire organization.

1. Plan your deployment
    1. Basic concepts
        1. Licensing
        2. Identity
        3. User management
        4. App deployment

5. Admin Console overview
6. Admin roles

2. Deployment Guides
    1. Named User deployment guide
    2. SDL deployment guide
    3. Deploy Adobe Acrobat
3. Deploy Creative Cloud for education
    1. Deployment guide
    2. Enable Adobe Express in Google Classroom
    3. Integration with Canvas LMS
    4. Integration with Blackboard Learn
    5. Configuring SSO for District Portals and LMSs
    6. Deploy Adobe Express through Google App Licensing
    7. Add users through Roster Sync
    8. Kivuto FAQ
    9. Primary and Secondary institution eligibility guidelines

2. Set up your organization
    1. Identity types | Overview
    2. Set up identity | Overview
    3. Set up organization with Enterprise ID
    4. Setup Azure AD federation and sync
        1. Set up SSO with Microsoft via Azure OIDC
        2. Add Azure Sync to your directory
        3. Azure Connector FAQ
    5. Set up Google Federation and sync
        1. Set up SSO with Google Federation
        2. Add Google Sync to your directory
        3. Google federation FAQ
    6. Set up organization with Microsoft ADFS

7. Deploy apps and updates
    1. Overview
        1. Deploy and deliver apps and updates
        2. Plan to deploy
        3. Prepare to deploy
    2. Create packages
        1. Package apps via the Admin Console
        2. Create Named User Licensing Packages
        3. Adobe templates for packages
        4. Manage packages
        5. Manage device licenses
        6. Serial number licensing
    3. Customize packages
        1. Customize the Creative Cloud desktop app
        2. Include extensions in your package
    4. Deploy Packages
        1. Deploy packages
        2. Deploy Adobe packages using Microsoft Intune
        3. Deploy Adobe packages with SCCM
        4. Deploy Adobe packages with ARD
        5. Install products in the Exceptions folder
        6. Uninstall Creative Cloud products
        7. Use Adobe provisioning toolkit enterprise edition
        8. Adobe Creative Cloud licensing identifiers
    5. Manage updates
        1. Change management for Adobe enterprise and teams customers
        2. Deploy updates
    6. Adobe Update Server Setup Tool (AUSST)
        1. AUSST Overview

        3. Manage invoices
        4. Change contract owner
        5. Change reseller
    2. Assign licenses to a Teams user
    3. Add products and licenses
    4. Renewals
        1. Teams membership: Renewals
        2. Enterprise in VIP: Renewals and compliance
    5. Automated expiration stages for ETLA contracts
    6. Switching contract types within an existing Adobe Admin Console
    7. Purchase Request compliance
    8. Value Incentive Plan (VIP) in China
    9. VIP Select help
9. Reports & logs
    1. Audit Log
    2. Assignment reports
    3. Content Logs
10. Get help

5.4 Analytics Reporting API

Comparison of Analytics APIs

API Type	Fully Processed	Real-Time	Livestream	Data Warehouse
Description	Fully-processed, finalized data that is available in all Analytics interfaces.	Partially-processed, limited metrics available within seconds of collection.	Partially-processed hit data available within seconds of collection.	Fully-processed, finalized data that is used for pulling large data exports.

API Type	Fully Processed	Real-Time	Livestream	Data Warehouse
Latency	30-90 minutes	Seconds - 10 minutes	Seconds - 10 minutes	90+ minutes
Processing Completion	Full	Partial	Partial	Full
Reporting Interfaces	Analysis Workspace, Reports & Analytics, Report Builder, API	Real-Time report in Reports & Analytics, Report Builder, 1.4 API	API only	Data Warehouse, API
Data granularity	Summarized	Summarized	Hit level	Summarized
Visitor Profile processing	Yes	No	No	Yes
Segment support	Yes	No	No	Partial

5.6 How processing rules work

Processing rules let you make changes to data based on defined conditions. When attributes or values match defined conditions, values can be set and deleted, and events can be set.

Processing rules are applied to data as it is collected, and rules are applied to all data that comes through the AppMeasurement libraries and through the Data Insertion API. Processing rules also apply to the full and log data sources. These sources contain data that represents a `hit` or an action that a user takes. Processing rules do not apply to other data sources.

Important Concepts

The following table contains key concepts you need to understand when using processing rules:

Concept	Details
Rules apply to a single report suite.	Copy processing rules to another report suite

Concept	Details
Processing rules are applied in the order listed.	If an action changes a value, subsequent conditions use the new value.
Processing rules are applied immediately to the report suite after they are saved.	Changes from processing rules should be visible in your report suite within minutes of saving. When testing processing rules, we recommend configuring real-time reports in your test report suite so you can quickly see the results of a processing rule.
Processing rules are the only way to access to context data variables.	Copy a Context Data Variable to an eVar
Processing rules are applied before VISTA rules and Marketing Channel rules.	Processing Order
Hits cannot be excluded.	You can use VISTA rules to exclude hits.
The product string, referrer, and user agent cannot be changed.	Referrer and user agent are read-only. The product string is not available.
Mobile device attributes and classifications are not available.	The mobile device lookup occurs before processing rules, but attributes are not available in processing rules.
Query string parameters cannot be read beyond the first 255 characters of a URL if you are running JavaScript AppMeasurement H.25.2 or earlier. JavaScript AppMeasurement H.25.3 and later provide the full URL including all query string parameters to processing rules.	Upgrade to H.25.3 or later, or read query string parameters from long URLs client-side and store values in Context Data variables.
Query string values must be encoded in Unicode or UTF-8 to be read by processing rules.	This might affect multibyte characters that are passed using query strings.
You are limited to 150 rules with 30 conditions each for each report suite.	Processing rule limits are per report suite, not per company.
Processing rules must be set up to retrieve context data variables before data is sent.	Processing rules are applied as server calls are sent. Values stored in context data variables are discarded if they are not copied using processing rules.

Concept	Details
Value comparisons in the UI are case insensitive.	Cleaning up Values in a Report .
Context data variable names can contain only alphanumeric characters, underscores and dots. Any additional characters are stripped out.	For example, The context data variable `login_page-home` automatically becomes `login_pagehome`. All data sent to the `login_page-home` variable is allocated under `login_pagehome`. Context data variables that contain unsupported characters cannot be added in the Processing Rules interface.
Caret (^) is a special character in the processing rules system.	To match a single caret character, use two caret characters (^^).

Processing Rule Conditions

Conditions check page variables for a matching value or if a value is present. Multiple conditions can be added and you can select if all conditions must be matched.

You can create a rule with no conditions to always execute defined actions.

Variables are not automatically checked for values before actions occur. For example, Prop1 contains a value of “something”, and eVar1 is empty. If you set Prop1 to equal eVar1 both values will be empty. If you need to avoid this add a condition to check for the presence of a value.

Processing Rule Actions

Actions set page variables, delete page variables, or trigger events. Actions can also concatenate values to display in a report.

For example, you might want to display `category:product` by concatenating two variables.