Introduction

This document describes the data schema used to send email-based information to a user's Everest account via the Everest Universal Endpoint (referred to simply as *endpoint*).

Note: *This document assumes you have created a "Universal Integration" under your Everest account, and you have the integration key associated with it. This is a 32-character identifier that you will use to send data to your Everest dashboard.*

Sending Data

To send your data to Everest, format it according to the specifications in this document, and send it via HTTP POST to `https://analytics.250ok.com/webhooks/{integrationKey}`, where `integrationKey` is obtained from the Universal Integration you created in your Everest account.

The endpoint will discard anything it doesn't recognize, so keep the following in mind to avoid common errors:

- Include a **Content-Type: application/json** header.
- Use **POST** over **HTTPS**.
- Send JSON-formatted data.
- Data is always sent in an array, whether one event or a thousand.
- String-serialize your data prior to sending (e.g. `JSON.stringify()`).

If you have any questions or your integration doesn't appear to be working, please contact support@validity.com for assistance. If possible, please include a link to a private Request Bin containing a few sample POSTs as you would send them to us.

Ensuring High Data Quality

It's important that you fully implement every recommended field so that we can do the best analysis of your data and present the most useful experience. However, because we recognize that each customer's ability to retain all that data will vary widely, we will cache certain pieces of information and use them to fill in missing fields. In order for us to do this, you must provide the fields ***at least once***, preferably in a `created` event so we have it as soon as possible. For more detail on this, please see the Common Fields section under Event Schemas.

For best results, we **strongly** recommend that you include the `messageId` field in all events. This is the single most useful value for identifying per-message metadata. If that is not possible, be sure to always include the `ip`, `to`, and `sendTime` fields.

Supported Events

The endpoint accepts data for the following events based on message or recipient activity.

Event Type	**Description**
`created`	Generated when a message is created and handed off to another system for sending. Some ESPs support this event, if you're sending through your own infrastructure this event is analogous to queuing the message to an SMTP server.
`delivered`	Generated when the intended recipient's domain has accepted the message.
`deferred`	Generated when the intended recipient's domain has temporarily refused delivery, as in the case of a connection or message quota being exceeded.
`filtered`	Generated when the sending system determines it will not attempt to send the message. An example would be because the recipient is on an internal suppression list.
`bounced`	Generated when the intended recipient's domain permanently refuses delivery for any reason.
`read`	Generated when a recipient has opened the message.
`click`	Generated when a link embedded in the message has been clicked.
`unsubscribed`	Generated when a recipient opts-out of some or all communications from the sender.
`complained`	Generated when an FBL report has been received.

Reserved Fields

This is the primary list of fields and their descriptions. Individual event specifications are provided in the sections that follow.

Field	Description
`attempts`	***Integer;*** The number of delivery attempts before the message was delivered or dropped.
`bounceCode`	***Integer;*** A numeric bounce classification, if available. The *bounce_class* field from SparkPost or the value returned by BounceStudio are good examples.
`deviceIP`	***String;*** For Read and Click events this is the IP address, in dotted-quad format, of the device on which the user opened the message or clicked a tracked link.
`dkimDomain`	***String;*** The domain associated with the DKIM signature whose validation results are recorded in the dkimResult field.
`dkimResult`	***String;*** The outcome of DKIM validation.
`event`	***String;*** The event being reported. See the previous section for a list of valid event names.
`eventTime`	***Integer;*** The time the event originally occurred, expressed in milliseconds since the epoch.
`fblDomain`	***String;*** The recipient domain reporting a spam complaint.
`from`	***String;*** The value of the From: header of the message associated with this event.
`ip`	***String;*** The IP address, in dotted-quad format, responsible for transmitting the message associated with this event.
`messageId`	***String;*** The value of the Message-ID header from the message associated with this event.
`method`	***String;*** For unsubscribed events, the mechanism used to unsubscribe. Either \`http\` or \`smtp\`.
`properties`	***Object;*** A collection of key/value pairs. Note that even if this field is not supplied, any non-reserved keys and their values will be collected during parsing and inserted into the properties object. Choose whichever method works best for you.
`sendTime`	***Integer;*** The time the message associated with this event was considered 'sent' by the user. Normally, this is the same as the eventTime of the created

	event, or the same as the message's Date: header. Like eventTime, this value must always be expressed as milliseconds from the epoch.
`smtpFrom`	***String;*** The value of the MAIL FROM command used when transmitting the message.
`smtpLog`	***String;*** The SMTP response received when a message is delivered, deferred, or bounced (rejected in protocol).
`smtpTo`	***String;*** The value of the RCPT TO command used when transmitting the message.
`spfDomain`	***String;*** The domain associated with the SPF result recorded in spfResult.
`spfResult`	***String;*** The outcome of SPF verification.
`subject`	***String;*** The value of the Subject: header of the message associated with this event.
`tags`	***String;*** An array of string values.
`to`	***String;*** The value of the To: header of the message associated with this event.
`url`	***String;*** For click events (and read, if you wish) the URL associated with the event. For clicks this would be the URL visited by the user. For reads this would be the URL used to track user activity (e.g. a tracking pixel or other form of beacon).
`userAgent`	***String;*** The value of the User-Agent: header of the HTTP request associated with the Read or Click event.

Event Schemas

Common Fields

These fields can be present in all events. As previously mentioned, we recognize that you may face technical challenges in doing this. So, for your convenience, we will cache these fields if we see them on the `created` or `delivered` events and will use the cached data to flesh out subsequent events omitting them.

Name	Required	Recommended	Cached	Notes
`event`	*			Unidentified or unrecognized events are discarded.
`eventTime`		*		If omitted, we will use the time you sent the event to us, which could create misleading event histories.
`from`		*	*	
`ip`		*	*	If omitted, prevents us from correlating IP-based reputation threats and delivery issues.
`messageId`	*			If omitted, we use other data to correlate events, but the method is fallible and can create misleading event histories/summaries.
`properties`			*	
`sendTime`		*	*	
`smtpFrom`		*	*	
`smtpTo`		*	*	
`subject`		*	*	If omitted, value of analytics tools is diminished, you will be unable to correlate campaign subject lines with performance information.
`tags`			*	
`to`		*	*	If omitted, value of analytics tools is diminished, you will be unable to correlate positive and negative

				deliverability and campaign performance data with individual recipient behaviors.

If it's no problem for you to send them on all events, great! If you can gain some efficiencies by doing it only once, we **strongly** recommend sending them on the created event. As that's the first logical event in the message/recipient interaction lifecycle, we can put your data to best use by receiving it on that event.

Event: Created

Generated when a message is created and handed off to another system for sending. Some ESPs support this event, if you're sending through your own infrastructure this event is analogous to queuing the message to an SMTP server.

Event Fields

Same as Common Fields, with `event` set to `created`.

Event: Delivered

Generated when the intended recipient's domain has accepted the message.

Event Fields

Incorporates all Common Fields, with `event` set to `delivered`, and adds the following unique fields:

- `attempts`

Event: Deferred

Generated when the intended recipient's domain has temporarily refused delivery, as in the case of a connection or message quota being exceeded.

Event Fields

Incorporates all Common Fields, with `event` set to `deferred`, and adds the following unique fields:

- `attempts`
- `smtpLog`

Event: Filtered

Generated when the sending system determines it will not attempt to send the message. An example would be because the recipient is on an internal suppression list.

Event Fields

Incorporates all Common Fields[1], with `event` set to `filtered`, and adds the following unique fields:

- `reason`

Event: Bounced

Generated when the intended recipient's domain permanently refuses delivery for any reason.

Event Fields

Incorporates all Common Fields, with `event` set to `bounced`, and adds the following unique fields:

- `bounceCode`
- `smtpLog`

Event: Read

Generated when a recipient has opened the message.

Event Fields

Incorporates all Common Fields, with `event` set to `read`, and adds the following unique fields:

- `deviceIP`
- `url`
- `userAgent`

Event: Click

Generated when a link embedded in the message has been clicked.

Event Fields

Incorporates all Common Fields, with `event` set to `click`, and adds the following unique fields:

- `deviceIP`
- `url`
- `userAgent`

---

[1] With the possible exception of *messageId* which may not yet have been generated. A case where the *filtered* event is generated AND *messageId* is present could be the case where the sender expires the message after several attempts to send it failed. Whether this case is represented as a *bounced* or *filtered* event is a matter of local policy. Bounces are factored into estimated impacts to reputation, so choose carefully.

Event: Unsubscribed

Generated when a recipient opts-out of some or all communications from the sender.

Event Fields

Incorporates all Common Fields, with `event` set to `unsubscribed`, and adds the following unique fields:

- `method`

Event: Complained

Generated when an FBL report has been received.

Event Fields

Incorporates all Common Fields, with `event` set to `click`, and adds the following unique fields:

- `fblDomain`

Event Example

This is an example of what a correctly serialized JSON payload containing a single event object would look like when ready for transmission to Everest[2].

```
[
     {

“event”: ”created”,
“eventTime”: 1502401894063,
“from”: ”Jon Snow <jon@castleblack.com>”,
“ip”: ”10.0.0.1”,
“messageId”: ”20170810-012345@raven.castleblack.com”, “properties”: {
“walkers”: true },
“sendTime”: 1502401892060,
“smtpFrom”: “bounces@castleblack.com”,
“smtpTo”: “sam@citadel.edu”,
“subject”: “how\’s gilly?”,
“tags”: [“ale”, “mutton”],
“to”: “Samuel Tarly <sam@citadel.edu>”

     }
]
```

[2] Spaces inserted for legibility.