Medio Systems Inc.
One Convention Place
701 Pike Street, Suite 1500

Seattle, WA 98101

Phone: 206.262.3700
Fax: 206.262.3799

Medio Platform
Data Collection Service
REST API Developer Guide

Version 2.0
August 2012

Table of Contents

1. Introduction

1.1 Overview

Medio Data Collection Service (DCS) is a standards-based web service that offers a single endpoint for clients to send data to the Medio platform. The service is designed to collect information about *events*. In general, an event describes something that happened at some point in time. A wide variety of data sources contain information about events. For example:

- Web server access logs
- Click-through logs
- Application-level events like user interactions

You can access the Medio DCS service at the following endpoint:

```
http[s]://events.medio.com/events/v2
```

DCS supports both HTTP and HTTPS protocols, however we recommend using HTTPS whenever possible.

1.2 Event Schemas

The DCS Service accepts data that conforms to a Medio-defined schema or to a schema that you specify. Please speak with your Medio account representative to discuss your event schema before sending data to the DCS Service.

2. REST API

The DCS REST API contains two primary resources: Append and Bulk Append. This section provides the API specification for these resources, outlines the API's authentication protocol, and specifies the data format to use when making requests to DCS.

1.3 API Keys and Authentication

Your Medio account representative will provide you with an API Key to send events to DCS. A DCS API key consists of a long string of characters that is specific to your organization and application. The API key influences how your application's data is displayed in your reports, so you should obtain a unique API key for each one of your applications.

A sample API Key looks as follows:

```
042429f548f54c5a4a5f5e5940425563014464f35a5d5d53547b5679612f3b377b3606701b297a26
183d277446253f3b272d2134da2c05011c1c1717391123122f50121f52
```

The key should be specified in the header `x-avalanche-api-key` with every request that you make to DCS. See Section 2.4 for more details on request headers.

1.4 Append Requests

Use the `append` resource to stream events to DCS in small batches. For example, if your application maintains a small buffer of events and frequently flushes them to DCS, you should use the `append` resource. The `append` resource allows you to send data of up to 1 megabyte in size (1048576 bytes) in a single request. If this size is exceeded, the service will reject the request with `HTTP 413 Request Entity Too Large` (see Section 2.8 for more details on HTTP response codes).

Use the `POST` HTTP method to send data to the `append` resource at the following URL:

```
http[s]://events.medio.com/events/v2/append
```

1.5 Bulk Append Requests

Use the `bulkappend` resource to send a large number of events to DCS in bulk. For example, if your application aggregates events over a long period of time and sends them infrequently to DCS, you should use the `bulkappend` resource. The `bulkappend` resource is identical in every way to the `append` resource except that it does not impose any limit on the size of the request data.

Use the `POST` HTTP method to send data to the `bulkappend` resource at the following URL:

```
http[s]://events.medio.com/events/v2/bulkappend
```

1.6 Request Headers

The following HTTP headers are required with every request:

- **Content-Type**
  The Content-Type header is required and dictates how the request body will be interpreted. You may append a charset specification to this header as defined by the HTTP 1.1 protocol specification, however the service assumes UTF-8 if no charset specification is provided. The following values for the header are supported and any other value will cause the request to fail with status `HTTP 415 Unsupported Media Type`:
  - `text/tsv` or `text/tab-separated-values`
  - `text/csv` (indicating comma-separated values)
- **Content-Length**
  The Content-Length header is required. If this header is missing, the DCS Service will respond with status `HTTP 411 Length Required` (see http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.12). If the value is larger than 1 megabyte (1048576 bytes), the DCS service will respond with status `HTTP 413 Request Entity Too Large` (see http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html#sec10.4.14). Note that if the payload is compressed (using the Content-Encoding header), the length of the compressed payload should be used here in compliance with the HTTP standard.
- **x-avalanche-date**
  The value of this header should be the date and time when the request was initiated in the client application. The value should be formatted in accordance with the HTTP Date header as described in http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.3.1. The x-avalanche-date header is not strictly required, however we strongly recommend that you include it in every request as it is used to determine the offset between the server's time and the client's reported time. If the header is not specified, the service will use the standard HTTP Date header in its place, however the Date header is not mandatory according to the HTTP standard.

The following HTTP headers are optional and you may omit them when not needed:

- **Content-Encoding**
  This header dictates how the DCS Service will interpret the request body. If you wish to compress the data that you send to DCS, set the value of this header to one of the following and the data will be interpreted according to the HTTP specification (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11):
  - `gzip`
  - `compress`
  - `deflate`
  - `bzip2`
- Connection
  The HTTP connection between the client and the server is not persistent by default, which is contrary to the default for the HTTP 1.1 protocol (http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.10). If you wish to reduce the overhead of establishing an HTTP connection with the service for continuous streaming, you may send the `Connection: Keep-Alive` header. If the service encounters this header it may keep the HTTP connection open for up to 15 seconds, however the service does not guarantee this behavior.

1.7 Request Body

The body of the `POST` request should be consistent with the values of the `Content-Type` and `Content-Encoding` headers.

DCS currently supports the TSV (tab-separated values) and CSV (comma-separated values) request body formats. You may compress the request body with one of the supported compression schemes as indicated by the `Content-Encoding` header.

The request body should consist of one or more event records. Each event record is parsed independently of other events in the request and one request may contain events of different types.

1.8 CSV/TSV Event Encoding

An event consists of a single CSV/TSV record, which is made up of a sequence of fields (values) separated by a delimiter. A well-formed event record should have at least two fields: and Event Type and an Event Timestamp. Any additional fields are considered part of the Event Payload.

The event type should abide by the restrictions and guidelines outlined in Appendix A and should be specified as the first field in the record. The event timestamp should be specified as the second field in the record and should be expressed as the number of milliseconds elapsed since 01/01/1970 00:00:00 UTC (not counting leap seconds). In other words, it should be expressed as a Unix Time with millisecond precision.

The following is an example of a valid event record (in CSV format):

```
PageView,1326706472435,78.98.97.34,http://www.medio.com/,http://google.com?q=predictive+analytics
```

1.9 Representing Values in CSV/TSV Format

The DCS Service supports data encoded in the CSV format as outlined in Section 2 of RFC 4180, with the following constraints relaxed:

3. A line feed alone (`LF`), a carriage return (`CR`) alone or the sequence `CRLF` may be used to delimit the end of a record.
4. Comments (lines that start with hash character #) and blank lines may be found anywhere and in any number in the body, not just in the first line.

The DCS Service also supports the TSV format as a variant of the CSV format where the separator is changed to a tab character rather than a comma character. All other provisions and requirements of RFC 4180 are taken to apply to TSV as well. Note that the tab character may not be contained in field values when using TSV format.

1.10 Responses

The DCS service communicates success and error conditions back to the calling client via the standard mechanisms defined by the HTTP standard. Clients should be designed to handle the error conditions documented below. The following table summarizes all the possible HTTP response codes and their interpretations:

HTTP Code	HTTP Message	Interpretation(s)	Notes
204	No Content	The request was well formed and the service accepted all events.	
200	Success	Partial success. The service rejected some events that were malformed.	The response body will contain further information about why the service rejected some events. See Section 2.9 more information.
400	Bad Request	This response code will be returned in the following situations: • The request was empty (has `Content-Length: 0`). • All events in the request were malformed and were rejected by the service. • There is a structural problem with the request that makes it an invalid HTTP request.	If the service rejects all the events in the request, the response body will contain further information about the rejected events. See Section 2.9 more information.
405	Method Not Allowed	Request has a method of `GET`, `PUT` or any method other than `POST`.	The DCS Service resources only allow the HTTP `POST` method.
411	Length Required	The `Content-Length` header was missing from the request.	
413	Request Entity Too Large	An `append` request has a `Content-Length` header value larger than 1048576 (1 megabyte).	
415	Unsupported Media Type	The `Content-Type` header is different than the supported values listed in Section 2.4.	
4xx	(Other client errors)	Other 400-series error codes may occur in various conditions, as	

		documented by the HTTP standard.	
500	Internal Server Error	The server encountered an internal error and the client should retry the request after backing off for a short period.	More details on how to handle this response are outlined in Section 2.10. We recommend that clients use an exponential backoff approach in retrying the request.
503	Service Unavailable	The service is overloaded or is currently undergoing maintenance.	The client should retry the request after backing off for a short period. We recommend that clients use an exponential backoff approach in retrying the request.
5xx	(Other server errors)	Other 500-series errors may occur in various conditions, as documented by the HTTP standard.	

Table 1: DCS Service HTTP Response Codes

1.11 Partial Success

If the service indicates that the request was only partially successful, it means that the service encountered problems in parsing or interpreting some events. In this case the service returns a success code (`HTTP 200 Success`) and includes a description of the errors in the following JSON format in the body of the response:

```
{    "failureType":"PARTIAL",
     "cause":"Some events were malformed.",
     "rejectedEvents":[{
         "index": <0-based index of event in body>,
         "cause": <error message detailing reason of rejection>
     },
     ...    ]}
```

The following is an example response for a partially successful request:

```
{    "failureType":"PARTIAL",
     "cause":"Some events were malformed.",
     "rejectedEvents":[{
         "index": 2,
         "cause": "Cannot parse event: Value [bad_value] for field [3] is not
valid."
     },
     ...    ]}
```

If the service rejects all events in a request, it will return response code `HTTP 400 Bad Request` and the body of the response will contain the details of why each event was rejected. For example:

```
{     "failureType":"COMPLETE",
      "cause":"Request contained no valid events.",
      "rejectedEvents":[{
          "index": <0-based index of event in body>,
          "cause": <error message detailing reason of rejection>
      },
      ...     ]}
```

1.12 Unexpected Server Errors

When a client receives an `HTTP 500` response code, it should behave as if the entire request has failed (none of the events have been received by DCS). After backing off for a short time, the client should retry to send the request again.

APPENDIX A: EVENT TYPE GUIDELINES

A.2 CONSTRAINTS FOR THE EVENT TYPE FIELD

The format for the event type field must conform to the following rules:

- It should be between 3 and 64 characters in length.
- It may be made up of letters, numbers, the period, the underscore character and a hyphen.
- It must begin with a letter.

Using regular expressions, the format of an event type can be validated using an expression similar to the following: `[a-zA-Z][a-zA-Z0-9._-]{2,63}`

A.3 GUIDELINES FOR CHOOSING VALUES FOR THE EVENT TYPE PARAMETER

Here are a few tips when thinking about how to name your events:

- Strive to make your event type names as short as possible without introducing ambiguity. Feel free to use abbreviations and acronyms.
- If you think that you might change the schema of an event in the future — by adding or removing key-value pairs — consider appending a version number to your event type name. For example, an event type named 'purchase' could be named 'purchase-1' to indicate that it follows version 1 of your schema. Using versions for event type names helps to avoid problems in processing or reporting on data that have different schemas.
- If you choose to append a version to your event type names, use a version that is independent of your product version. The version is used to indicate the schema of your event, which you may want to keep the same across multiple versions of your product.