Campaign Commander

Email & Mobile Edition

Campaign Commander™ Email & Mobile Edition Cloud Service APIs

Document name	Data Mass Update REST API Guide
Service	Data management for mass updates and insertions
Protocol	REST over HTTP
Version	10.8
Last updated on	May 20, 2013

emailvision

Table of Contents

Introduction

About This Document

This document is a reference document for using Campaign Commander™ Email & Mobile Edition APIs. It does not explain the purpose or functions of Campaign Commander™ Email & Mobile Edition features. For information on these features, please consult the *Campaign Commander™ Online Help* or the *Campaign Commander™ Email & Mobile Edition User Guide*.

This document is intended for developers and project managers.

About Campaign Commander™ Email & Mobile Edition APIs

An Application Programming Interface (API) is a source code interface that a computer system or program library provides in order to support requests for services made from another computer program.

The goal of Campaign Commander™ Email & Mobile Edition APIs is to offer customers the ability to pilot a complete campaign from their own system.

Example: The introduction of a new promotional article in the e-commerce back office should automatically generate a new Campaign Commander™ Email & Mobile Edition campaign to targeted recipients.

Feedback

The Data Mass Update REST API Guide is constantly being enhanced to provide you with more and more information on using Campaign Commander™ Email & Mobile Edition API methods.

If you can't find the information you need or want to provide feedback, simply drop us a line at documentation@emailvision.com.

We look forward to hearing from you!

Support Options

Emailvision provides you with a dedicated Account Manager to accompany you throughout the execution of your projects in Campaign Commander™ Email & Mobile Edition. Your Account Manager is the gateway to support, training, and professional services. Working with your Account Manager, you can rely on Emailvision's deliverability and technical support teams for complex troubleshooting and optimization.

Training Options

Emailvision provides fully comprehensive training ranging from basic product training through to advanced modules and both strategic and tactical marketing courses. The training courses are designed to help you increase productivity, develop new methods, and share best practices to optimize your email marketing campaigns.

The Emailvision Academy, located in central London and in Paris, has state-of-the-art training rooms and equipment.

To get more information on training, go to the Emailvision Academy website: http://www.emailvisionacademy.co.uk

Useful Links

- Information on Emailvision’s products and services: http://www.emailvision.com
- Training: http://www.emailvisionacademy.co.uk

Disclaimer

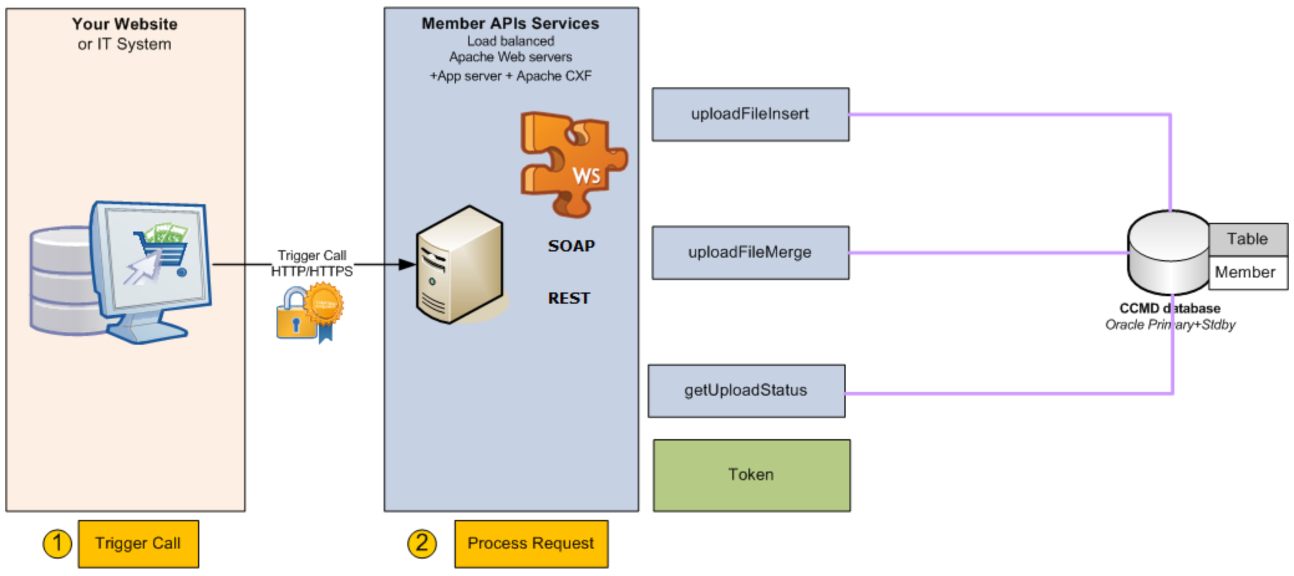
Introducing Campaign Commander APIs

Module Overview

The Data Mass Update module allows you to create and update your customers' profiles in your Campaign Commander™ Email & Mobile Edition member table.

This API offers you a seamless way to insert or merge a file of members through one single API call whenever you want and with the member fields you desire. You can upload up to 5 files at a time, each with a maximum size of 256 MB.

The member table is a single table stored in Emailvision's datacenter. It contains all the profile information of your recipients, such as email address, first name, last name, and any column defined during the life of your account.



Overview of the Data Mass Update API

The Data Mass Update API allows you to upload and update mailing lists in your Campaign Commander™ Email & Mobile Edition member list.

For further information on uploading and updating mailing lists, please consult the *Campaign Commander™ Email & Mobile Edition User Guide* or *Campaign Commander™ Online Help*.

The following methods are available:

Method	Description
Open Connection	This method provides a session token when given valid credentials.
Close API Connection	This method terminates the session token.
Upload a File and Insert the Members	This method uploads a file containing members and inserts them into the member table. Note: The maximum file size is 256 Mb. You can upload up to five files simultaneously per Campaign Commander™ account.
Upload a File and Merge the Members with the Existing Members	This method uploads a file containing members and merges them with those in the member table. Note: The maximum file size is 256 Mb. You can upload up to five files simultaneously per Campaign Commander™ account.
Get Last Upload	This method retrieves the last 20 uploads for the Campaign Commander™ Email & Mobile Edition account and their statuses.
Get Upload Status	This method retrieves the status of a file upload.
Get Log File	This method retrieves the log file associated with an upload.
Get Bad File	This method retrieves the lines of an uploaded file that could not be uploaded due to errors.

Data Mass Update API Use Cases

Upload a Mailing List into the Member List

To upload the member data of a mailing list into the member list:

1. Use the Open Connection method to open the connection.
2. Use the Upload a File and Insert the Members method to upload the mailing list file.
3. Use the Get Upload Status method to obtain the status of the upload.
4. Use the Get Log File method to see the details of what was and was not inserted into the member list.
5. Use the Close Connection method to close the connection.

Update the Member List

To update the member data in the member list:

1. Use the Open Connection method to open the connection.
2. Use the Upload a File and Merge the Members with the Existing Members method to update the member list with the data in the mailing list file.
3. Use the Get Upload Status method to obtain the status of the upload.
4. Use the Get Log File method to see the details of what was and was not updated in the member list.
5. Use the Close Connection method to close the connection.

Getting Started with Integration

Prerequisites

To access Campaign Commander™ Email & Mobile Edition’s APIs and take full advantage of this software’s ease of integration with other systems, you will need the following:

- An Internet connection
- A recent browser and operating system
- An active Campaign Commander™ Email & Mobile Edition account with the API feature enabled

Quick Start

The process for interfacing your website, CRM, or any other internal system with the APIs is quite straightforward.

Step 1: Get your API key in Campaign Commander™ Email & Mobile Edition

Note: You must have a dedicated API login. This login will NOT have access to Campaign Commander™ Email & Mobile Edition. Contact your Account Manager to have a dedicated API login.

To connect through the API, the user must first obtain a manager key using the CCMD Web Application.

Calling the connect method (with the login, password, manager key) will provide a token, to be used in all subsequent calls.

This token will expire in the following cases:

- When a close connection call is made.
- When the maximum number of calls per session, defined by the manager in Campaign Commander™ Email & Mobile Edition, is reached.
- When the session times out.

Step 2: Build your application

Integration Using APIs

The first step in getting started with web services is to configure the range of remote servers that will access this module.

Webmasters and developers should be able to interface with this new API with any programming language that uses standard HTTP calls.

List of APIs that are available:

- RESTful API
- SOAP API (see the *Data Mass Update SOAP API Guide*)

RESTful API

Description: RESTful API is the most standard way to remotely call a Web Service. REST requests are always sent over the HTTP protocol and can vary in format and methods of submission.

This API method is available in HTTP GET Path Info (PI).

It differs in the way parameters are organized in the URL. In the PI method, parameters are organized like a path.
<u>The order of all parameters is very important.</u>

- The path is composed of a series of values.
- The values are each separated by a forward slash sign (/).
- Below is an Internet browser URL location bar showing a URL with the Path Info:

http://en.wikipedia.org/index.php/param1/param2/param3

- API call summary:

HTTP GET (Path Info)
Submission URL `http://{server}apibatchmember/services/rest/batchmemberservice/{token}/batchmember/`
Parameters & associated values • All parameter names are case-sensitive. • When specific values are expected, it should be assumed that parameter values are case sensitive. • The order of parameters must be strictly followed.
Example call: `http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/getLastUpload`

URL Encoding Considerations (for HTTP GET methods only)

Some characters cannot be part of a URL - for example, spaces are not allowed. Some characters have a special meaning in a URL—for example, the hash (#) character is used to locate a specific point within a page, and the equals (=) character is used to separate a name from a value. A query string may need to be converted to satisfy these constraints. This can be done using a schema known as URL encoding. In particular, encoding the query string uses the following rules:

- Letters (A-Z and a-z) and numbers (0-9) are not encoded.
- The period (.), comma (,) , tilde (~), and underscore (_) characters are not encoded.
- A space is encoded as **%20**.
- The forward slash (/) is encoded as **%2F**.
- All other characters are encoded as **%FF** hex representation with any non-ASCII characters first encoded as UTF-8 (or other specified encoding).
- To encode as RFC 1738, use the + sign to replace spaces.

Security

As web services are accessible over the Internet and can be interfaced with any system, there is a risk of fraudulent access and usage of the system. To tighten the security, Campaign Commander™ Email & Mobile Edition APIs can be accessed using the HTTPS protocol coupled with the use of encrypted and matching tags.

To use HTTPS, just replace **HTTP** with **HTTPS** in all the submission URLs.

Connection

Prerequisite: To use Campaign Commander™ APIs, you need to have the API manager login provided by Emailvision and the associated password.

To connect through the API, you must first retrieve the manager key from Campaign Commander™ Email & Mobile Edition.

1. Go to **Account Administration** and select **Logins**.
2. Click the **Edit** icon next to your API manager.
3. In the **API** section of the popup window, copy the API key (also known as the manager key) and use it to open a connection to retrieve the token that will be used in your calls.

Calling the connect method (with the login, password, manager key) will provide a token, to be used in all subsequent calls.

This token will expire in the following cases:

- When a close connection call is made.
- When the maximum number of calls per session, defined by the manager in Campaign Commander™ Email & Mobile Edition, is reached.
- When the session times out.

Open Connection

This method provides a session token when given valid credentials.

WSDL Location

```
http://{server}/apibatchmember/services/BatchMemberService?wsdl
```

Note: Ask your Account Manager for your server name.

Input parameters **Required parameters**	**Description**	**Output parameters**	**Description**
login	The login provided for API access	**return**	The token to use in all other API calls
pwd	The password		
key	The manager key copied from Campaign Commander™ Email & Mobile Edition (see Connection on page 11)		

Error messages
You must fill in the apiName parameter to check rights of client on this API.
You must fill in the login parameter to authentifiate on this API.
You must fill in the password parameter to authentifiate on this API.
You must fill in the managerKey parameter to authentifiate on this API.
Error while decoding managerKey.
Your login is not valid !!
Your password is not valid !!
No manager retrieved for those login, password.
No available connection for manager {0}.
{0} doesn't exist or is not activated on client account.
{0} is not activated for the client.
This manager does not have authorized access to this API.
Error while parsing validDate on managerKey.
Date not valid on managerKey!
The managerKey is no longer valid. Your API access is closed!

REST Example

Input

REST QS

```
http://{server}/apibatchmember/services/rest/connect/open?login={login}&password=
{password}&key={key}
```

REST PI

```
http://{server}/apibatchmember/services/rest/connect/open/{login}/{password}/
{key}
```

Output

```
<response responseStatus="success">
   <result xsi:type="xs:string">
       {token}
   </result>
</response>
```

Close API Connection

This method terminates the session token.

REST URL

```
http://{server}/apibatchmember/services/rest/connect/close/{token}
```

Note: Ask your Account Manager for your server name.

Input parameter **Required parameters**	**Description**	**Output parameter**	**Description**
token	The connection token	**return**	The connection is closed if the operation was successful, otherwise an error code appears.

Error messages
You must fill in the token parameter
No available connection for the specified token.
An error occured on the server

REST Example

Input

```
http://{server}/apibatchmember/services/rest/connect/close/{token}
```

Output

```
<response responseStatus="success">
  <result xsi:type="xs:string">connection closed</result>
</response>
```

Upload a File and Insert the Members

This method uploads a file containing members and inserts them into the member table.

Note: The maximum file size is 256 Mb. You can upload up to five files simultaneously per Campaign Commander™ account.

This is a PUT method.

REST URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
insertUpload
```

Note: Ask your Account Manager for your server name.

Description

Members from the uploaded file will be inserted into the member table.

The **autoMapping** parameter uses the first line of the file as a mapping. If set to false or omitted the mapping has to be defined in the parameters of the mapping envelope (e.g. column 1 <=> EMAIL, column 2 <=> FIRSTNAME).

The data patterns used for the **dateFormat** parameter are:

- yyyy = Year
- MM = Month
- dd = Day
- HH = Hours (1 -12)
- HH24 = Hours (00-23)
- mi = Minutes (and not mm)
- ss = seconds
- XXX = time zone

Note: The data patterns of the date format are not case sensitive. They can be divided by spaces, backslashes (/), colons (:), and hyphens (-).

For example:

- dd/MM/yyyy
- dd/MM/yyyy HH:mi
- MM/dd/yyyy HH24:mi
- yyyy/MM/dd HH24:mi:ss
- dd/MM/yyyy HH:mmXXX
- MM/dd/yyyy HH:mmXXX
- yyyy/MM/dd HH:mmXXX
- dd-MM-yyyy HH:mmXXX
- MM-dd-yyyy HH:mmXXX
- yyyy-MM-dd HH:mmXXX
- dd/MM/yyyy HH:mm:ssXXX
- MM/dd/yyyy HH:mm:ssXXX
- yyyy/MM/dd HH:mm:ssXXX
- dd-MM-yyyy HH:mm:ssXXX

- MM-dd-yyyy HH:mm:ssXXX
- yyyy-MM-dd HH:mm:ssXXX

You can also use the option to deduplicate members in the file before inserting them:

- **dedup**: deduplication envelope parameter
- **criteria**: member field(s) to use as match criteria (e.g. LOWER(EMAIL))
- **order**: **first** or **last** -> the first or last occurrence of a duplicate entry in the file will be used

If you want to skip unsubscribed or quarantine members, **skipUnsubAndHBQ** must be set to **true**.

Input parameter **Required parameters**	**Description**	**Output parameters**	**Description**
token	The connection token	return	**uploadId** - The ID of the upload job
file	The content ID of the attachment to upload or the Base64-encoded file content.		
insertUpload	The upload configuration envelope containing the parameters defining the upload.		
fileName	The name of the file to upload		
fileEncoding	The encoding of the file (the default value is UTF-8)		
separator	The separator used: • comma (,) • semi-colon (;) • pipe (|) • tab		
skipFirstLine	Skips the first line in the file (default value is **false**)		
dateFormat	The date format used in the columns containing dates		
autoMapping	Set to **true** to automatically map the column headings in the file to the column headings in the database (default value is **false**).		
Deduplication Parameters			
dedup	The deduplication envelope parameter which must include **criteria**, **order**, and **skipUnsubAndHBQ** (optional). If not given, deduplication will be set to false.		
criteria	The field to use as the duplication key, e.g. LOWER (EMAIL)		
order	Set to **first** to keep the first occurrence of a duplicate entry in the file. Set to **last** to keep the last occurrence. Default value is **first**.		

Input parameter **Required parameters**	**Description**	**Output parameters**	**Description**
skipUnsubAndHBQ	Set to **true** to skip unsubscribed and quarantined members. Omit or set to **false** to include them.		
Mapping Parameters			
mapping	The mapping envelope parameter containing the column mapping definitions. Required if **autoMapping** is **false**.		
column	The column envelope parameter for the mapping parameters: **colNum**, **fieldName**, **dateFormat** (optional), and **defaultValue** (optional).		
colNum	The number of the column in the file. Required if **autoMapping** is **false**.		
fieldName	The column name in the database that should be linked to the specified file column number. Required if **autoMapping** is **false**.		
dateFormat	The date format that will override the date format set for the file for the specified column.		
defaultValue	If defined, the values contained in the column for the **colNum** of the file will be ignored and the defined default value will be inserted.		

Error messages
You must fill in the token parameter
Element 'mapping' is required.
Element 'fileName' is required.
Element 'separator' is required.
Too many uploads are still pending. The limit is 5 uploads by client. You must wait until some uploads are processed.
Element 'separator' is invalid. Allowed are: [, ; | tab].
Upload file is too big. The limit is 256 Mbytes.
Upload file is empty.
An error occured on the server

REST Example

Input

URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
insertUpload
```

Content-type

```
multipart/form-data; boundary=XyXyXy
```

Body

```
--XyXyXy
Content-Disposition: form-data; name="insertUpload";
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<insertUpload>
    <fileName>member.csv</fileName>
    <fileEncoding>UTF-8</fileEncoding>
    <separator>,</separator>
    <dateFormat>mm/dd/yyyy</dateFormat>
    <mapping>
        <column>
            <colNum>1</colNum>
            <fieldName>EMAIL</fieldName>
        </column>
        <column>
            <colNum>2</colNum>
            <fieldName>FIRSTNAME</fieldName>
        </column>
        <column>
            <colNum>3</colNum>
            <fieldName>DATEOFBIRTH</fieldName>
        </column>
    </mapping>
</insertUpload>
--XyXyXy--
--XyXyXy
Content-Disposition: form-data; name="inputStream";
filename="member.csv"
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

dG90b0Bs-
dXIudGVzdC5jb20sVG90by-
wwNi8xNS8xOTYzCnRhdGFAbHVyLnRlc3QuY29tLFRhdGEsMDIvMDMvMTk4MApib3Vib3VAbHVyLnRlc3QuY29tLEJvd
--XyXyXy--
```

Output

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<response responseStatus="success">
    <result xsi:type="xs:long" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">197323</result>
</response>
```

Upload a File and Merge the Members with the Existing Members

This method uploads a file containing members and merges them with those in the member table.

Note: The maximum file size is 256 Mb. You can upload up to five files simultaneously per Campaign Commander™ account.

This is a PUT method.

REST URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
mergeUpload
```

Note: Ask your Account Manager for your server name.

Description

Members from the uploaded file will be merged with those in the member table.

- The merge criteria must be specified as a comma-separated list of member fields (these must be defined in the mapping). You may even specify SQL functions.

  Examples:

  ```
  <criteria>FIRSTNAME,LASTNAME</criteria>

  <criteria>LOWER(EMAIL)</criteria>
  ```

- Only the columns that are to be replaced should be defined in the mapping envelopes with the **toReplace** parameter. Other columns will be ignored.
- The data patterns used for the **dateFormat** parameter are:
  - yyyy = Year
  - MM = Month
  - dd = Day
  - HH = Hours (1 -12)
  - HH24 = Hours (00-23)
  - mi = Minutes (and not mm)
  - ss = seconds
  - XXX = time zone

  Note: The data patterns of the date format are not case sensitive. They can be divided by spaces, backslashes (/), colons (:), and hyphens (-).

  For example:

  - dd/MM/yyyy
  - dd/MM/yyyy HH:mi
  - MM/dd/yyyy HH24:mi
  - yyyy/MM/dd HH24:mi:ss
  - dd/MM/yyyy HH:mmXXX
  - MM/dd/yyyy HH:mmXXX
  - yyyy/MM/dd HH:mmXXX
  - dd-MM-yyyy HH:mmXXX

- MM-dd-yyyy HH:mmXXX
- yyyy-MM-dd HH:mmXXX
- dd/MM/yyyy HH:mm:ssXXX
- MM/dd/yyyy HH:mm:ssXXX
- yyyy/MM/dd HH:mm:ssXXX
- dd-MM-yyyy HH:mm:ssXXX
- MM-dd-yyyy HH:mm:ssXXX
- yyyy-MM-dd HH:mm:ssXXX

Input parameter **Required parameters**	**Description**	**Output parameters**	**Description**
token	The connection token	return	**uploadId** - The ID of the upload job
file	The content ID of the attachment to upload or the Base64-encoded file content.		
mergeUpload	The upload configuration envelope containing the parameters defining the upload.		
fileName	The name of the file to upload		
fileEncoding	The encoding of the file (the default value is UTF-8)		
separator	The separator used: • comma (,) • semi-colon (;) • pipe (|) • tab		
skipFirstLine	Skips the first line in the file (default value is **false**)		
dateFormat	The date format used in the columns containing dates		
criteria	The field to use as merge criteria, e.g. LOWER(EMAIL)		
Mapping Parameters			
mapping	The mapping envelope parameter containing the column mapping definitions.		
colNum	The number of the column in the file		
fieldName	The column name in the database that should be linked to the specified file column number.		
toReplace	Defines whether a field value should or should not be replaced (default value is **false**). It must be present for the field value to be replaced.		
dateFormat	The date format that will override the date format set for the file for the specified column.		
defaultValue	If defined, the values contained in the column for the **colNum** of the file will be ignored and the defined default value will be inserted.		

Error messages
You must fill in the token parameter

Error messages
Element 'criteria' is required for merge
Element 'mapping' is required.
Element 'fileName' is required.
Element 'separator' is required.
Too many uploads are still pending. The limit is 5 uploads by client. You must wait until some uploads are processed.
Element 'separator' is invalid. Allowed are: [, ; | tab].
Upload file is too big. The limit is 256 Mbytes.
Upload file is empty.
An error occured on the server

REST Example

Input

URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
mergeUpload
```

Content-type

```
 multipart/form-data; boundary=XyXyXy
```

Body

```
--XyXyXy
Content-Disposition: form-data; name="mergeUpload";
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
<mergeUpload>
    <fileName>member.csv</fileName>
    <fileEncoding>UTF-8</fileEncoding>
    <separator>,</separator>
    <dateFormat>mm/dd/yyyy</dateFormat>
    <criteria>LOWER(EMAIL)</criteria>
    <mapping>
        <column>
            <colNum>1</colNum>
            <fieldName>EMAIL</fieldName>
        </column>
        <column>
            <colNum>2</colNum>
            <fieldName>FIRSTNAME</fieldName>
            <toReplace>true</toReplace>
        </column>
        <column>
            <colNum>3</colNum>
            <fieldName>DATEOFBIRTH</fieldName>
            <toReplace>true</toReplace>
        </column>
        <column>
            <fieldName>SEGMENT</fieldName>
            <defaultValue>TESTAPI</defaultValue>
        </column>
    </mapping>
</mergeUpload>
--XyXyXy--
--XyXyXy
Content-Disposition: form-data; name="inputStream";
filename="member.csv"
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

dG90b0Bs-
dXIudGVzdC5jb20sVG90by-
wwNi8xNS8xOTYzCnRhdGFAbHVyLnRlc3QuY29tLFRhdGEsMDIvMDMvMTk4MApib
3Vib3VAbHVyLnRlc3QuY29tLEJvdWJvdSwwNy8xOC8xOTcyCg==
--XyXyXy--
```

Output

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<response responseStatus="success">
    <result xsi:type="xs:long" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">197325</result>
</response>
```

Get Last Upload

This method retrieves the last 20 uploads for the Campaign Commander™ Email & Mobile Edition account and their statuses.

This is a GET method.

REST URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
getLastUpload
```

Note: Ask your Account Manager for your server name.

Description

For each of the 20 retrieved uploads, the following information is provided:

- **id**: The ID of the upload.
- **source**: The source application where the upload was created (API_BATCH_MEMBER or CCMD)
- **status**: The status of the upload:
  - **Pending status**
    - STORAGE: Upload file has been successfully uploaded and saved
    - VALIDATED: Upload file has been successfully validated
    - QUEUED: Upload has been queued for processing
    - IMPORTING: Database import is starting
  - **Error status**
    - ERROR: File validation has failed
    - FAILURE: Database import has failed
  - **Final status**
    - DONE: Upload is complete
    - DONE WITH ERRORS: Upload is complete but there were some errors

Input parameter **Required parameters**	**Description**	**Output parameters**	**Description**
token	The connection token	return	**uploads** - a list of the last 20 uploads

Error messages
You must fill in the token parameter
An error occured on the server

REST Example

Input

URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
getLastUpload
```

Content-type

```
text/xml
```

Output

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<response responseStatus="success">
    <lastUploads>
        <id>197325</id>
        <source>API_BATCH_MEMBER</source>
        <status>QUEUED</status>
    </lastUploads><lastUploads><id>197324</id><source>API_BATCH_MEMBER</-
source><status>QUEUED</status>
    </lastUploads>
    <lastUploads>
        <id>197323</id>
        <source>API_BATCH_MEMBER</source>
        <status>QUEUED</status>
    </lastUploads>
    <lastUploads>
        <id>197315</id>
        <source>API_BATCH_MEMBER</source>
        <status>DONE</status>
    </lastUploads>
</response>
```

Get Upload Status

This method retrieves the status of a file upload.

This is a GET method.

REST URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
{uploadId}/getUploadStatus
```

Note: Ask your Account Manager for your server name.

Description

The process of an upload job is divided into several steps. Each step is associated with a specific status.

This method provides the current status of an upload job.

Pending status

- STORAGE: Upload file has been successfully uploaded and saved
- VALIDATED: Upload file has been successfully validated
- QUEUED: Upload has been queued for processing
- IMPORTING: Database import is starting

Error status

- ERROR: File validation has failed
- FAILURE: Database import has failed

Final status

- DONE: Upload is complete
- DONE WITH ERRORS: Upload is complete but there were some errors

Input parameter **Required parameters**	**Description**	**Output parameters**	**Description**
token	The connection token	return	**status** - the status of the upload job
uploadId	The ID of the upload job		

Error messages
You must fill in the token parameter
You must fill in the uploadId parameter.
No flatUpload found !
An error occured on the server

REST Example

Input

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
{uploadId}/getUploadStatus
```

Output

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<response responseStatus="success">
    <uploadStatus>
        <status>QUEUED</status>
    </uploadStatus>
</response>
```

Get Log File

This method retrieves the log file associated with an upload.

REST

This is a GET method.

REST URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
{uploadId}/getLogFile
```

Note: Ask your Account Manager for your server name.

Input parameter **Required parameters**	**Description**	**Output parameters**	**Description**
token	The connection token	return	The logs for the upload
uploadId	The ID of the upload job		

Error messages
You must fill in the token parameter
No flatUpload found!
An error occured on the server

REST Example

Input

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
{uploadId}/getLogFile
```

Output

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?><response respon-
seStatus="success"><result xsi:type="xs:string" xmlns:x-
si="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
SQL*Loader: Release 10.2.0.1.0 - Production on Tue Jun 21 15:04:49 2011

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

Control File:   /var/emv/DATA_UPLOAD/35882/LUR_ACCOUNT_197254.ctl
Character Set UTF8 specified for all input.
Using character length semantics.

Data File:      /var/emv/DATA_UPLOAD/35882/LUR_ACCOUNT_197254.dat
  Bad File:     /var/emv/DATA_UPLOAD/35882/LUR_ACCOUNT_197254.bad
  Discard File:  none specified

 (Allow all discards)
```

```
Number to load: ALL
Number to skip: 1
Errors allowed: 4
Bind array:     64 rows, maximum of 256000 bytes
Continuation:    none specified
Path used:      Conventional

Table TEMP_FU_197254, loaded from every logical record.
Insert option in effect for this table: APPEND
TRAILING NULLCOLS option in effect

   Column Name                  Position   Len  Term Encl Datatype
------------------------------ ---------- ----- ---- ---- ---------------------
CLIENT_ID                                                 CONSTANT
    Value is '35882'
EMAIL                               FIRST   765   ;  O(") CHARACTER
    SQL string for column : "DECODE(REGEXP_INSTR(:EMAIL,-
'.+@.+..+',1),0,NULL,LOWER(TRIM(:EMAIL)))"
TEMPORARY_MEMBER_ID                  NEXT     *   ;  O(") CHARACTER
    SQL string for column : "DECODE(:TEMPORARY_MEMBER_ID,null, SEQ_TMP_197254.nex-
tval, SEQ_TMP_197254.nextval)"


Table TEMP_FU_197254:
  3 Rows successfully loaded.
  0 Rows not loaded due to data errors.
  0 Rows not loaded because all WHEN clauses were failed.
  0 Rows not loaded because all fields were null.


Space allocated for bind array:                 66048 bytes(64 rows)
Read   buffer bytes: 1048576

Total logical records skipped:          1
Total logical records read:             3
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Tue Jun 21 15:04:49 2011
Run ended on Tue Jun 21 15:04:49 2011

Elapsed time was:     00:00:00.08
CPU time was:         00:00:00.01
</result>
</response>
```

Get Bad File

This method retrieves the lines of an uploaded file that could not be uploaded due to errors.

This is a GET method.

REST URL

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
{uploadId}/getBadFile
```

Note: Ask your Account Manager for your server name.

Input parameter Required parameters	Description	Output parameters	Description
token	The connection token	return	The lines that could not be uploaded
uploadId	The ID of the upload job		

Error messages
You must fill in the token parameter
No flatUpload found!
An error occured on the server

REST Example

Input

```
http://{server}/apibatchmember/services/rest/batchmemberservice/{token}/batch-
member/
{uploadId}/getBadFile
```

Output

```
<response responseStatus="success">
<result xsi:type="xs:string">ccommanderqa@hotmail.co.uk      bob16   sinclar16
   33600000015     16/02/2010^@0906/07/1984        QA
          ccommanderqa3@yahoo.com.au      bob26   sinclar26       33600000025
26/08/2010^@0916/02/1973        QA
          testm30@wanadoo.fr      bob17   sinclar17       33600000016
17/02/2010      08/04/1 87      QA
          ccommanderqa@fastmail.co.uk     bob27   sinclar27       33600000026
27/08/2010^@0917/06/1972        QA
</result></response>
```

Upload Errors

When an error occurs, the response will contain an object that contains 3 elements:

- status: INVALID_FILE, INVALID_PARAMETERS, MAX_NB_UPLOADS, INVALID_UPLOAD_ID, GET_MANAGER_FAILED, GET_STATUS_FAILED, GET_CLIENT_FAILED, CREATE_JOB_FAILED
- description: short description of the problem
- fields: parameters that are the source of the problem

Parameter Errors	
INVALID_FILE	• Upload file is empty • Upload file is too big. The limit is 250 M bytes.
INVALID_ PARAMETERS	• Element 'mapping' is required. • Element 'fileName' is required. • Element 'separator' is required. • Element 'separator' is invalid. Allowed are: [, ; | tab] • Element 'criteria' is required for merge
MAX_NB_UPLOADS	Too many uploads are still pending. You must wait until some uploads are processed.

Server Errors	
GET_MANAGER_ FAILED	An error happened on the server while retrieving the manager from the database.
GET_STATUS_ FAILED	An error happened on the server while retrieving the status from the database.
GET_CLIENT_FAILED	An error happened on the server while retrieving the client from the database.
CREATE_JOB_FAILED	An error happened on the server while saving the upload job to the database.

Example:

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
  <response responseStatus="failed">
    <description>No flatUpload found!</description>
    <fields>12345</fields>
    <status>GET_FLATUPLOAD_FAILED</status>
  </response>
```

Upload Examples

PHP Examples

Upload a File and Insert the Members

```
<?php

class EMVPlatform
{
        const T1 = "emvapi.emv2.com";
        const T2 = "p2apic.emv2.com";
        const T3 = "p3apic.emv2.com";
        const T4 = "p4apic.emv2.com";
        const T5 = "p5apic.emv2.com";
        const T6 = "p6apic.emv2.com";
        const E1 = "emvapi.emv3.com";
        const E2 = "p2apie.emv3.com";
        const E3 = "p3apie.emv3.com";
        const E4 = "p4apie.emv3.com";
        const E5 = "p5apie.emv3.com";
}

class EMVAPIError
{
        public function __construct($title,$description,$message)
        {
                $output = <<<EOC
        <div style="padding:10px;font-family:Arial,Verdana;width:600px;margin:0
auto 0 auto; border:1px solid #CC0000; background-color:#FFF3F3; color:#CC0000;
margin-top:60px">
                <span style="font-size:12px; font-weight:bold">Emailvision API
Error</span><br>
                <span style="font-size:25px; font-weight:bold">$title</span><br>
                <hr noshade>
                <span style="font-size:16px">$description</font>
        </div>
EOC;
                echo $output;
                exit;
        }
}

class EMVAPIStreamMethod      { const PUT     = 100; const GET      = 101;  }
class EMVAPIStreamRequestType { const REGULAR = 200; const UPSTREAM = 201;  }

class EMVAPIMergeUploadParams
{
        public $fileName;
        public $fileEncoding;
        public $separator;
        public $dateFormat;
        public $criteria;
        public $skipFirstLine;
        public $mapping = array();
        public $_fileContent;

        public function __loadFile()
```

```
       {
               $this->_fileContent = base64_encode(file_get_contents($this->
filePath.$this->fileName));
       }
}

class EMVMassUpdateMapping
{
       var $column;
}

class EMVAPIStream
{
       private $boundarySeed;

       public function __construct()
       {
               $this->generateBoundarySeed();
       }

       private function generateBoundarySeed()           { $this->boundarySeed =
md5(uniqid("5AFF8C33A03")); }
       private function assembleHeader() { return "Content-Type: multipart/
form-data; boundary=".$this->boundarySeed; }

       private function assembleUpstreamBody($parameters)
       {
               $parameters->__loadFile();

               $output  = "--".$this->boundarySeed."\r\n";
               $output .= <<<EOC
Content-Disposition: form-data; name="mergeUpload";
Content-Type: text/xml

<?xml version="1.0" encoding="UTF-8"?>
       <mergeUpload>
               <criteria>{$parameters->criteria}</criteria>
               <fileName>{$parameters->fileName}</fileName>
               <separator>{$parameters->separator}</separator>
               <fileEncoding>{$parameters->fileEncoding}</fileEncoding>
               <skipFirstLine>{$parameters->skipFirstLine}</skipFirstLine>
               <dateFormat>{$parameters->dateFormat}</dateFormat>
               <mapping>
EOC;

               foreach ($parameters->mapping as $column)
               {
                       $output .= <<<EOC
       <column>
               <colNum>{$column['colNum']}</colNum>
               <fieldName>{$column['fieldName']}</fieldName>
       </column>
EOC;
               }

               $output .= <<<EOC
                       </mapping>
               </mergeUpload>
EOC;
```

```
                $output .= "\r\n--".$this->boundarySeed."--\r\n";
                $output .= "--".$this->boundarySeed."\r\n";
                $output .= <<<EOC
Content-Disposition: form-data; name="inputStream"; filename="{$parameters->
fileName}"
Content-Type: application/octet-stream
Content-Transfer-Encoding: base64

EOC;
                $output .= "\r\n".$parameters->_fileContent;
                $output .= "\r\n--".$this->boundarySeed."--\r\n";

                return $output;
        }

        public function request($url,$method,$type,$content=null)
        {
                $parameters = array();

                if ($type == EMVAPIStreamRequestType::UPSTREAM)
                {
                        $parameters = array(
                                "http" => array
                                (
                                        "method"  => "PUT",
                                        "header"  => $this->assembleHeader(),
                                        "content" => $this->assembleUpstreamBody
($content)
                                )
                        );

                        print_r($parameters);
                }
                else if ($type == EMVAPIStreamRequestType::REGULAR)
                {
                        $parameters = array(
                                "http" => array(
                                        "method"                => "GET",
                                        "max_redirects" => 0,
                                        "ignore_errors" => 1
                                )
                        );
                }

                $context = stream_context_create($parameters);
                $stream  = fopen($url,"r",false,$context);

                $output  = new EMVAPIStreamResponse();
                $output->metadata = stream_get_meta_data($stream);
                $output->content  = stream_get_contents($stream);

                // Check for errors
                if (isset($output->metadata['wrapper_data'][0]) && $output->
metadata['wrapper_data'][0] === "HTTP/1.1 500 Internal Server Error")
                {
                        new EMVAPIError($response->description,$response->
status,$output);
                }
                fclose($stream);

                return simplexml_load_string($output->content);
        }
}

class EMVAPIStreamResponse { public $metadata; public $content; }

class EMVMassUpdate
{
        private $login;
        private $platform;
        private $password;
        private $key;
        private $token;
        private $_remoteMethod;
        private $_parameters = array();
        private $_stream;

        public function __construct($platform,$login,$password,$key)
        {
                $this->platform = $platform;
                $this->login    = $login;
                $this->password = $password;
                $this->key      = $key;
                $this->_stream  = new EMVAPIStream();

                $this->openConnection();
        }

        public function __destruct()
        {
                $this->closeConnection();
        }

        private function getUrlEndpoint($remoteMethod,$parameters=null)
        {
                $output = "http://".$this->plat-
form."/apibatchmember/services/rest/".$remoteMethod;

                if ($parameters != null) $output .= "?".$this->
serializeQueryParameters($parameters);

                return $output;
        }

        private function serializeQueryParameters($input)
        {
                return urldecode(http_build_query($input));
        }

        private function openConnection()
        {
                // $method,$type,$content=null
                $url = $this->getUrlEndpoint("connect/open",array(
                        "login"    => $this->login,
                        "password" => $this->password,
                        "key"      => $this->key
```

```
               ));

               $response        = $this->_stream->request($url,
EMVAPIStreamMethod::GET, EMVAPIStreamRequestType::REGULAR);
               $this->token = $response->result[0];
       }

       private function closeConnection()
       {
               $url = $this->getUrlEndpoint("connect/close/".$this->token);
               $this->_stream->request($url,EMVAPIStreamMethod::GET,
EMVAPIStreamRequestType::REGULAR);
       }

       public function mergeUpload(EMVAPIMergeUploadParams $parameters)
       {
               $url = $this->getUrlEndpoint("batchmemberservice/".$this->token."/
batchmember/mergeUpload");
               $this->_stream->request($url,EMVAPIStreamMethod::PUT,
EMVAPIStreamRequestType::UPSTREAM, $parameters);
       }
}


// Use the EMVPlatform according to your account's Pod. In case you don't have
this information,
// please ask the EMV Staff.
$client = new EMVMassUpdate(EMVPlatform::T5,"YOUR_LOGIN","YOUR_PASSWORD","YOUR_
KEY");

$params = new EMVAPIMergeUploadParams();
$params->filePath      = "C:\\my\\path\\tofile\\"; // In case it's unix, it can
also be used as /my/path/to/file/
$params->fileName      = "test.txt";
$params->fileEncoding  = "UTF-8";
$params->separator     = "|";
$params->dateFormat    = "dd/mm/YYYY";
$params->criteria      = "LOWER(EMAIL)";
$params->skipFirstLine = "false";
$params->mapping = array
(
       array("colNum" => 1, "fieldName" => "EMAIL")
);

$client->mergeUpload($params);


?>
```

Reference

WADL

The Web Application Description Language (WADL) is a machine-readable XML-based language that provides a model for describing HTTP-based web applications (such as REST web services).

Web Services

The W3C defines a Web service as a software system designed to support interoperable Machine to Machine interaction over a network. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate XML messages that follow the SOAP-standard. Common in both the field and the terminology is the assumption that there is also a machine readable description of the operations supported by the server, a description in the WSDL. The latter is not a requirement of SOAP endpoint, but it is a prerequisite for automated client-side code generation in the mainstream Java and .NET SOAP frameworks. Some industry organizations, such as the WS-I, mandate both SOAP and WSDL in their definition of a Web service.

WSDL

The Web Services Description Language (WSDL, pronounced 'wiz-dull' or spelled out, 'W-S-D-L') is an XML-based language that provides a model for describing Web services. Version 2.1 has not been endorsed by the World Wide Web Consortium (W3C). Version 2.0, for which several drafts have been released, is expected to become a W3C recommendation. WSDL is an XML-based service description on how to communicate using web services. The WSDL defines services as collections of network endpoints, or ports. WSDL specification provides an XML format for documents for this purpose. WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special datatypes used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL.

XML

The Extensible Markup Language (XML) is a W3C-recommended general-purpose markup language. The XML recommendation specifies both the structure of XML, and the requirements for XML processors. XML is considered "general-purpose" because it enables anyone to originate and use a markup language for many types of applications and problem domains. Numerous formally defined markup languages are based on XML, such as RSS, MathML, GraphML, XHTML, Scalable Vector Graphics, MusicXML, and thousands of others. XML's primary purpose is to facilitate the sharing of data across different information systems, particularly systems connected via the Internet. It is a simplified subset of Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible.