Bulk API 2.0

Version 41.0, Winter '18

18

@salesforcedocs

Last updated: November 2, 2017

CONTENTS

CHAPTER 1 Bulk API 2.0

In this chapter ...

Bulk API 2.0 provides a simple interface for quickly loading large amounts of data into your Salesforce org.

Bulk API 2.0 includes the following features.

Support for OAuth 2.0
Bulk API 2.0 supports all OAuth flows supported by Salesforce's other REST APIs.

Automatic File Batching
Bulk API 2.0 simplifies uploading large amounts of data by breaking the data into batches automatically. All you have to do is upload a CSV file with your record data and check back when the results are ready.

Daily Limits Based on Number of Records
Instead of limiting the amount of data uploaded daily by the quantity of jobs, Bulk API 2.0 uses a limit of total records uploaded. The limit is 100 million records per 24-hour period.

Process for Creating Bulk API 2.0 Requests

Use the Bulk API 2.0 to process large amounts of data by defining a job, providing data for that job, and monitoring the results.

Create a new Bulk API 2.0 request by doing the following steps.

1. Create a job. The job defines a specific Bulk API 2.0 operation, such as inserting new records, or updating records.
2. Upload data for the job. For data sets under 20,000 characters, upload the data as part of a multipart request when you create the job. For larger data sets, up to 150MB (after base64 encoding), upload the data by creating an upload data request. Don't delete your local CSV data until you've confirmed that Salesforce has processed all records successfully.
3. If you're using a separate request to upload data, once you're done uploading data, close the job by setting the job state to `UploadComplete`. Salesforce will start processing the job at this point.
4. Check the status of the job. When a job's state is `JobComplete`, you can retrieve its results.
5. Retrieve the results of the completed job by checking the successful results, failed results, and unprocessed records.

How Requests Are Processed

Salesforce processes Bulk API 2.0 jobs you create and uses job state to describe the status of a queued job.

Salesforce queues a new job for processing once you've created it, uploaded data for the job, and moved the job to the `UploadComplete` state. Once the job is de-queued, Salesforce starts processing the job operation and moves the job to the `InProgress` state. Once the job has been fully processed, Salesforce moves the job to the `JobComplete` state, or `Failed` state, depending on if the job was successfully processed or not. The following table summarizes the different job states for Bulk API 2.0.

State of Job	Description
Open	The job was created and is open for data uploads.
UploadComplete	All job data has been uploaded and the job is ready to be processed. Salesforce has put the job in the queue.
InProgress	The job is being processed by Salesforce. This includes automatic optimized chunking of job data and processing of job operations.
JobComplete	The job was processed.
Failed	The job could not be processed successfully.
Aborted	The job was canceled by the job creator, or by a user with the "Manage Data Integrations" permission.

Salesforce automatically chunks the data for your job into multiple internal batches to improve performance while processing the job. Salesforce creates a separate batch for each 10,000 records in your job data, up to a daily maximum of 100 million records. If the limit is exceeded while processing your job data, the remaining data isn't processed.

For each internal batch, if Salesforce can't process all the records in a batch in 10 minutes or less, the batch fails. Salesforce will re-try failed batches up to a maximum of 10 retries. If the batch still can't be processed after 10 retries, the entire job is moved to the `Failed` state and remaining job data is not processed. Use the Failed Record Results and Unprocessed Record Records resources to determine what records were not processed and what errors occurred.

CHAPTER 2 Examples

In this chapter ...

Use the following examples to understand how to do tasks with Bulk API v2.0

Walkthrough for Bulk Insert

This walkthrough guides you through the steps for creating a job to insert new records, uploading data for the job, checking status, and retrieving the results.

- Read through all the steps in this walkthrough. You might also want to review the rest of this document to get a quick overview of Bulk API 2.0 calls and concepts.
- Familiarize yourself with workbench. This walkthrough uses workbench to issue Bulk API 2.0 calls, but you can use any tool or development environment that can make REST requests.

1. Create a CSV file containing your data.

   Using your preferred spreadsheet tool, create a CSV file that contains the records you want to insert. The first row of the CSV file lists the field names for the object you're working with. Each subsequent row corresponds to a record that you want to insert.

   For a simple example list of Contacts that can be used to create a CSV file, see: Bulk API Sample Contact Data

   For additional information on preparing CSV files, such as delimiter options and valid date/time formats, see Data Files.

2. Create a job.

   To do any Bulk API 2.0 task, such as inserting or updating records, you first create a job. The job specifies the type of object, such as Contact, that you're loading. The job also specifies the operation you're performing, such as insert or delete. Once you've created a job, you can use the resulting job ID in Bulk API 2.0 requests to close or abort (cancel) the job.

   a. In workbench, log in to your org.

   b. Go to **Utilities** > **REST Explorer**. Use Bulk API 2.0 to create a job by issuing a POST request with the following details:

      Request Headers:

      ```
      Content-Type: application/json; charset=UTF-8
      Accept: application/json
      ```

      URI:

      ```
      /services/data/vXX.X/jobs/ingest/
      ```

      Request Body:

      ```
      {
          "object" : "Contact",
          "contentType" : "CSV",
          "operation" : "insert"
      }
      ```

      If you're inserting an object other than Contact, replace “Contact” in the request body with your object name.

      You should get a response that includes the job ID, with a job state of `Open`. You'll use the job ID in Bulk API 2.0 calls in the next steps. You'll also use the URL in the `contentUrl` field of the response in the next step when you upload your data.

3. Upload your CSV data.

   After creating a job, you're ready to upload your data. You provide record data using the CSV file you created earlier.

   In workbench's REST Explorer, use Bulk API 2.0 to create a job data request with the CSV data. Issue a PUT request with the following details:

Request Headers:

```
Content-Type: text/csv
Accept: application/json
```

URI:

Use the URI provided in the `contentUrl` field of the response from step 1. The URI has a format similar to the following example.

```
/services/data/vXX.X/jobs/ingest/JOB ID/batches/
```

Request Body:

```
(Content of your CSV file)
```

For the Request Body, copy and paste the CSV data into workbench's Request Body text field. With Bulk API 2.0, you can submit CSV data that does not in total exceed 150 MB in size (after base64 encoding).

You should get a response that includes the job ID still in the `Open` state. You could cancel the job at this point by using a PATCH to set the job state to `Aborted`.

4. Close the job.

   Once you're done submitting data, you can inform Salesforce that the job is ready for processing by closing the job.

   In workbench's REST Explorer, issue a PATCH request with the following details:

   Request Headers:

   ```
   Content-Type: application/json; charset=UTF-8
   Accept: application/json
   ```

   URI:

   ```
   /services/data/vXX.X/jobs/ingest/JOB ID/
   ```

   Request Body:

   ```
   {
       "state" : "UploadComplete"
   }
   ```

5. Check the job status and results.

   To get basic status information on a job, such as the overall job state or the number of records processed, use a GET request with the following details:

   Request Headers:

   ```
   Content-Type: application/json; charset=UTF-8
   Accept: application/json
   ```

   URI:

   ```
   /services/data/vXX.X/jobs/ingest/JOB ID/
   ```

   Once a job has been completed and is in the `JobComplete` state (or `Failed` state), you can get details of which job data records were successfully processed by issuing a GET request with the following details:

Request Headers:

```
Content-Type: application/json; charset=UTF-8
Accept: text/csv
```

```
/services/data/vXX.X/jobs/ingest/JOB ID/successfulResults/
```

You get a response that contains CSV data, with each row containing a record ID and information on whether that record was successfully processed or not. To get details on records that encountered an error during processing, use a GET request using the `failedResults` resource instead of the `successfulResults` resource.

Walkthrough for Bulk Insert With A Multipart Request

This walkthrough guides you through the steps for creating a job to insert new records, creating batches for the job, checking status, and retrieving the results. This walkthrough uses a single multipart request to create the job and upload job data.

- Read through all the steps in this walkthrough. You might also want to review the rest of this document to get a quick overview of Bulk API 2.0 calls and concepts.
- Familiarize yourself with workbench. This walkthrough uses workbench to issue Bulk API 2.0 calls, but you can use any tool or development environment that can make REST requests.

1. Create a CSV file containing your data.

   Using your preferred spreadsheet tool, create a CSV file that contains the records you want to insert. The first row of the CSV file lists the field names for the object you're working with. Each subsequent row corresponds to a record that you want to insert.

   For a very simple example list of Contacts that can be used to create a CSV file, see: Bulk API Sample Contact Data

   For additional information on preparing CSV files, such as delimiter options and valid date/time formats, see Data Files.

   For this walkthrough, since we'll use a multipart request to create the job and upload the data, the total size of the CSV data should not exceed 20,000 characters.

2. Create a job.

   To do any Bulk API 2.0 task, such as inserting or updating records, you first create a job. The job specifies the type of object, such as Contact, that you're loading. The job also specifies the operation you're performing, such as insert or delete. In this walkthrough we'll also include job data in the request that creates the job.

   Once you've created a job, you can use the resulting job ID in Bulk API 2.0 requests to close or abort (cancel) the job.

   a. In workbench, log in to your org.

   b. Go to **Utilities** > **REST Explorer**. Use Bulk API 2.0 to create a new job and upload job data by issuing a POST request with the following details:

      Request Headers:

      ```
      Content-Type: multipart/form-data; boundary=BOUNDARY
      Accept: application/json
      ```

      URI:

      ```
      /services/data/vXX.X/jobs/ingest/
      ```

      Request Body:

      ```
      --BOUNDARY
      Content-Type: application/json
      ```

```
Content-Disposition: form-data; name="job"

{
  "object":"Contact",
  "contentType":"CSV",
  "operation": "insert"
}

--BOUNDARY
Content-Type: text/csv
Content-Disposition: form-data; name="content"; filename="content"

(Content of your CSV file)
--BOUNDARY--
```

If you're inserting an object other than Contact, replace "Contact" in the request body with your object name. In this example "BOUNDARY" is used to mark the request body boundaries between the job details and the job CSV data.

You should get a response that includes the job ID, with a job state of `UploadComplete`. You'll use the job ID in Bulk API 2.0 calls in the next steps.

3. Check the job status and results.

   To get basic status information on a job, such as the overall job state or the number of records processed, use a GET request with the following details:

   Request Headers:

   ```
   Content-Type: application/json; charset=UTF-8
   Accept: application/json
   ```

   URI:

   ```
   /services/data/vXX.X/jobs/ingest/JOB ID/
   ```

   Once a job has been completed and is in the `JobComplete` state (or `Failed` state), you can get details of which job data records were successfully processed by issuing a GET request with the following details:

   Request Headers:

   ```
   Content-Type: application/json; charset=UTF-8
   Accept: text/csv
   ```

   ```
   /services/data/vXX.X/jobs/ingest/JOB ID/successfulResults/
   ```

   You'll get a response that contains CSV data, with each row containing a record ID and information on whether that record was successfully processed or not. To get details on records that encountered an error during processing, use a GET request using the `failedResults` resource instead of the `successfulResults` resource.

Walkthrough for Bulk Upsert

This walkthrough guides you through the steps for creating a job to upsert records, uploading data for the job, checking status, and retrieving the results.

- Read through all the steps in this walkthrough. You might also want to review the rest of this document to get a quick overview of Bulk API 2.0 calls and concepts.

- Familiarize yourself with workbench. This walkthrough uses workbench to issue Bulk API 2.0 calls, but you can use any tool or development environment that can make REST requests.

1. Confirm that your object is using an external ID field.

   Upserting records requires an external ID field on the object involved in the job. This field is used to identify existing records in the org. Bulk API 2.0 uses the external ID field to decide if a record in the bulk job data will be used to update an existing record, or create a new record.

2. Create a CSV file containing your data.

   Using your preferred spreadsheet tool, create a CSV file that contains the records you want to upsert. The first row of the CSV file lists the field names for the object you're working with. Each subsequent row corresponds to a record that you want to insert.

   One of the columns in the CSV must correspond to the external ID field identified in step 1.

   For additional information on preparing CSV files, such as delimiter options and valid date/time formats, see Data Files.

3. Create a job.

   To do any Bulk API 2.0 task, such as inserting or updating records, you first create a job. The job specifies the type of object, such as Contact, that you're loading. The job also specifies the operation you're performing, such as insert or delete. Once you've created a job, you can use the resulting job ID in Bulk API 2.0 requests to close or abort (cancel) the job.

   a. In workbench, log in to your org.

   b. Go to **Utilities** > **REST Explorer**. Use Bulk API 2.0 to create a new job by issuing a POST request with the following details:

      Request Headers:

      ```
      Content-Type: application/json; charset=UTF-8
      Accept: application/json
      ```

      URI:

      ```
      /services/data/v***XX.X***/jobs/ingest/
      ```

      Request Body:

      ```
      {
          "object" : "Contact",
          "externalIdFieldName" : ***name of external id field***
          "contentType" : "CSV",
          "operation" : "upsert"
      }
      ```

      Note that you need to provide the name of the external ID field when you create the upsert job, using the `externalIdFieldName` property. If you're inserting an object other than Contact, replace “Contact” in the request body with your object name.

      You should get a response that includes the job ID, with a job state of `Open`. You'll use the job ID in Bulk API 2.0 calls in the next steps. You'll also use the URL in the `contentUrl` field of the response in the next step when you upload your data.

4. Upload your CSV data.

   After creating a new job, you're ready to upload your data. You'll provide the record data using the CSV file you created earlier.

   In workbench's REST Explorer, use Bulk API 2.0 to create a new job data request with the CSV data. Issue a PUT request with the following details:

Request Headers:

```
Content-Type: text/csv
Accept: application/json
```

URI:

Use the URI provided in the `contentUrl` field of the response from step 3. The URI will have a format similar to the following example.

```
/services/data/vXX.X/jobs/ingest/JOB ID/batches/
```

Request Body:

```
(Content of your CSV file)
```

For the Request Body, copy and paste the CSV data into workbench's Request Body text field. With Bulk API 2.0, you can submit CSV data that does not in total exceed 150MB in size (after base64 encoding).

You should get a response that includes the job ID still in the `Open` state. You could cancel the job at this point by using a PATCH to set the job state to `Aborted`.

5. Close the job.

   Once you're done submitting data, you can inform Salesforce that the job is ready for processing by closing the job.

   In workbench's REST Explorer, issue a PATCH request with the following details:

   Request Headers:

   ```
   Content-Type: application/json; charset=UTF-8
   Accept: application/json
   ```

   URI:

   ```
   /services/data/vXX.X/jobs/ingest/JOB ID/
   ```

   Request Body:

   ```
   {
       "state" : "UploadComplete"
   }
   ```

6. Check the job status and results.

   To get basic status information on a job, such as the overall job state or the number of records processed, use a GET request with the following details:

   Request Headers:

   ```
   Content-Type: application/json; charset=UTF-8
   Accept: application/json
   ```

   URI:

   ```
   /services/data/vXX.X/jobs/ingest/JOB ID/
   ```

   Once a job has been completed and is in the `JobComplete` state (or `Failed` state), you can get details of which job data records were successfully processed by issuing a GET request with the following details:

Request Headers:

```
Content-Type: application/json; charset=UTF-8
Accept: text/csv
```

```
/services/data/vXX.X/jobs/ingest/JOB ID/successfulResults/
```

You'll get a response that contains CSV data, with each row containing a record ID and information on whether that record was successfully processed or not. To determine which records were created and which records were updated during the upsert job, examine the `Created` field of the successful results. To get details on records that encountered an error during processing, use a GET request using the `failedResults` resource instead of the `successfulResults` resource.

CHAPTER 3 Data Files

In this chapter ...

Use comma-separated value (CSV) data when submitting data rows for Bulk API 2.0 jobs. Bulk API 2.0 supports several formatting options with CSV data, such as supporting multiple field delimiter characters and line ending characters.

Prepare CSV Files

The first row in a CSV file lists the field names for the object that you're processing. Each subsequent row corresponds to a record in Salesforce.

All the records in a CSV file must be for the same object. You specify this object in the job associated with the batch.

Note the following when working with CSV files with Bulk API 2.0:

- You must include all required fields when you create a record. You can optionally include any other field for the object.
- If you're updating a record, any fields that aren't defined in the CSV file are ignored during the update.
- Files must be in UTF-8 format. Files are converted to base64 when received by Salesforce. This conversion can increase the data size by approximately 50%. To account for the base64 conversion increase, upload data that does not exceed 100 MB.
- Bulk API 2.0 supports several field delimiter characters: backquote (`), caret (^), comma, pipe (|), semicolon, and tab. The default delimiter is comma. Specify the delimiter to use when you create your job, using the `columnDelimiter` request field.
- Bulk API 2.0 supports several line ending formats: linefeed, and carriage-return plus linefeed. The default line ending is linefeed. Specify the line ending to use when you create your job, using the `lineEnding` request field.
- Use double-quotes to escape characters in field values that would otherwise get interpreted as field delimiters or line endings. For example, if a field value includes a comma, and comma is the current column delimiter for the job, you must wrap the field value in double-quotes in the CSV data, like “Director, Marketing”.
- Field values aren't trimmed. A space before or after a delimiter is included in the field value. A space before or after a double quote generates an error for the row. For example, `John,Smith` is valid; `John, Smith` is valid, but the second value is `" Smith"`; `."John", "Smith"` is not valid.
- Empty field values are ignored when you update records. To set a field value to `null`, use a field value of `#N/A`.
- Fields with a `double` data type can include fractional values. Values can be stored in scientific notation if the number is large enough (or, for negative numbers, small enough), as indicated by the W3C XML Schema Part 2: Datatypes Second Edition specification.

Valid Date Format in Records

Use the *`yyyy-MM-ddTHH:mm:ss.SSS+/-HHmm`* or *`yyyy-MM-ddTHH:mm:ss.SSSZ`* formats to specify `dateTime` fields.

- `yyyy` is the four-digit year
- `MM` is the two-digit month (01-12)
- `dd` is the two-digit day (01-31)
- 'T' is a separator indicating that time-of-day follows
- `HH` is the two-digit hour (00-23)
- `mm` is the two-digit minute (00-59)
- `ss` is the two-digit seconds (00-59)
- `SSS` is the optional three-digit milliseconds (000-999)
- `+/-HHmm` is the Zulu (UTC) time zone offset
- 'Z' is the reference UTC timezone

When a timezone is added to a UTC dateTime, the result is the date and time in that timezone. For example, 2002-10-10T12:00:00+05:00 is 2002-10-10T07:00:00Z and 2002-10-10T00:00:00+05:00 is 2002-10-09T19:00:00Z. See W3C XML Schema Part 2: DateTime Datatype.

Use the *`yyyy-MM-dd+/-HHmm`* or *`yyyy-MM-ddZ`* formats to specify `date` fields. See W3C XML Schema Part 2: Date Datatype.

Relationship Fields in a Header Row

Many objects in Salesforce are related to other objects. For example, Account is a parent of Contact. You can add a reference to a related object in a CSV file by representing the relationship in a column header. When you're processing records in the Bulk API, you use ***RelationshipName.IndexedFieldName*** syntax in a CSV column header to describe the relationship between an object and its parent, where *RelationshipName* is the relationship name of the field and *IndexedFieldName* is the indexed field name that uniquely identifies the parent record. Use the `describeSObjects()` call in the SOAP-based SOAP API to get the `relationshipName` property value for a field.

Some objects also have relationships to themselves. For example, the `Reports To` field for a contact is a reference to another contact. If you're inserting a contact, you could use a `ReportsTo.Email` column header to indicate that you're using a contact's `Email` field to uniquely identify the `Reports To` field for a contact. The `ReportsTo` portion of the column header is the `relationshipName` property value for the `Reports To` field. The following CSV file uses a relationship:

```
FirstName,LastName,ReportsTo.Email
Tom,Jones,buyer@salesforcesample.com
```

Note the following when referencing relationships in CSV header rows:

- You can use a child-to-parent relationship, but you can't use a parent-to-child relationship.
- You can use a child-to-parent relationship, but you can't extend it to use a child-to-parent-grandparent relationship.
- You can only use indexed fields on the parent object. A custom field is indexed if its `External ID` field is selected. A standard field is indexed if its `idLookup` property is set to `true`. See the Field Properties column in the field table for each standard object.

Relationship Fields for Custom Objects

Custom objects use custom fields to track relationships between objects. Use the relationship name, which ends in `__r` (underscore-underscore-r), to represent a relationship between two custom objects. You can add a reference to a related object by representing the relationship in a column header.

If the child object has a custom field with an `API Name` of `Mother_Of_Child__c` that points to a parent custom object and the parent object has a field with an `API Name` of `External_ID__c`, use the column header `Mother_Of_Child__r.External_ID__c` to indicate that you're using the parent object's `External ID` field to uniquely identify the `Mother Of Child` field. To use a relationship name in a column header, replace the `__c` in the child object's custom field with `__r`. For more information about relationships, see Understanding Relationship Names in the *Salesforce SOQL and SOSL Reference Guide* at `www.salesforce.com/us/developer/docs/soql_sosl/index.htm`.

The following CSV file uses a relationship:

```
Name,Mother_Of_Child__r.External_ID__c
CustomObject1,123456
```

Relationships for Polymorphic Fields

A polymorphic field can refer to more than one type of object as a parent. For example, either a contact or a lead can be the parent of a task. In other words, the `WhoId` field of a task can contain the ID of either a contact or a lead. Since a polymorphic field is more flexible, the syntax for the column header has an extra element to define the type of the parent object. The syntax is ***ObjectType:RelationshipName.IndexedFieldName***. The following sample includes two reference fields:

1. The `WhoId` field is polymorphic and has a `relationshipName` of `Who`. It refers to a lead and the indexed `Email` field uniquely identifies the parent record.

2. The `OwnerId` field is not polymorphic and has a `relationshipName` of `Owner`. It refers to a user and the indexed `Id` field uniquely identifies the parent record.

```
Subject,Priority,Status,Lead:Who.Email,Owner.Id
Test Bulk API polymorphic reference field,Normal,Not
Started,lead@salesforcesample.com,005D0000001AXYz
```

Warning: The ***ObjectType:*** portion of a field column header is only required for a polymorphic field. You get an error if you omit this syntax for a polymorphic field. You also get an error if you include this syntax for a field that is not polymorphic.

Sample CSV Files

The following examples demonstrate different ways to use CSV data with Bulk API 2.0.

For simplicity, all the following examples only show a few records.

Simple CSV

This example contains three Account records and specifies the Name, Description, and NumberOfEmployees fields for each record.

```
Name,Description,NumberOfEmployees
TestAccount1,Description of TestAccount1,30
TestAccount2,Another description,40
TestAccount3,Yet another description,50
```

A job that uses this CSV data might be defined with the following job properties.

```
{
  "object" : "Account",
  "contentType" : "CSV",
  "operation" : "insert"
}
```

CSV with Alternate Line Ending

This example contains two Contact records and specifies three fields for each record. The data was created on a Windows platform, and each line ends with a carriage return and line feed. The carriage return is displayed as "^M" in this example.

```
FirstName,LastName,Description^M
Tom,Jones,Branding guru^M
Ian,Dury,Fuzzy logic expert^M
```

A job that uses this CSV data and specifies that carriage return + line feed is used as the line ending sequence would use the following job properties.

```
{
  "object" : "Contact",
  "contentType" : "CSV",
  "operation" : "insert",
  "lineEnding" : "CRLF"
}
```

CSV with Semicolon Delimiter and Escaped Fields

This example contains two Contact records and specifies five fields for each record. The field delimiter is a semicolon instead of a comma. The Description fields contain characters that must be escaped using double-quotes, including a line-break in the second record.

```
FirstName;LastName;Title;Birthdate;Description
Tom;Jones;Senior Director;1940-06-07Z;"Self-described as ""the top"" branding guru"
Ian;Dury;Chief Imagineer;1965-12-11Z;"Expert in fuzzy logic design; Knowledgeable in AI
Influential in technology purchases."
```

A job that uses this CSV data and specifies that semicolon is used as the column delimiter would use the following job properties.

```
{
  "object" : "Contact",
  "contentType" : "CSV",
  "operation" : "insert",
  "columnDelimiter" : "SEMICOLON"
}
```

CSV with Relationship Field

This example contains two Contact records and specifies FirstName, LastName, and Owner.Email fields for each record. This example assumes a unique User record exists that has an Email value of "mfellow@salesforce.com" and creates a relationship with this record and the Contact records. If the User record doesn't exist, or if there are multiple User records with an Email value of "mfellow@salesforce.com", the relationship can't be created and the job fails.

```
FirstName,LastName,Owner.Email
Joe,User,mfellow@salesforce.com
Jane,User,mfellow@salesforce.com
```

A job that uses this CSV data might be defined with the following job properties.

```
{
  "object" : "Contact",
  "contentType" : "CSV",
  "operation" : "insert"
}
```

CSV for Upsert Using External IDs

This example contains three Contact records and specifies FirstName, LastName, Phone, and ExternalId__c for each record. This example assumes the custom ExternalId__c external ID field has been added to Contact.

```
FirstName,LastName,Phone,ExternalId__c
Mark,Brown,4155558787,"1001"
Dave,Stillman,4155552212,"1002"
Joe,Smith,2125556363,"5001"
```

A job that uses this CSV data might be an upsert job defined with the following properties.

```
{
  "object" : "Contact",
  "externalIdFieldName" : "ExternalId__c",
  "contentType" : "CSV",
```

```
  "operation" : "upsert"
}
```

CHAPTER 4 Reference

In this chapter ...

The API reference for Bulk API 2.0 includes all the actions that you can perform with jobs.

Create a Job

Creates a job, which represents a bulk operation (and associated data) that is sent to Salesforce for asynchronous processing. Provide job data via an **Upload Job Data** request, or as part of a multipart create job request.

URI

/services/data/v*XX.X*/jobs/ingest

Formats

JSON

HTTP Method

POST

Authentication

`Authorization: Bearer token`

Parameters

None.

Request Body

Property	Type	Description	Required or Optional
`columnDelimiter`	ColumnDelimiterEnum	The column delimiter used for CSV job data. The default value is `COMMA`. Valid values are: • `BACKQUOTE`—backquote character (`) • `CARET`—caret character (^) • `COMMA`—comma character (,) which is the default delimiter • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character	Optional
`contentType`	ContentType	The content type for the job. The only valid value (and the default) is `CSV`.	Optional
`externalIdFieldName`	string	The external ID field in the object being updated. Only needed for upsert operations. Field values must also exist in CSV job data.	Required for upsert operations
`lineEnding`	LineEndingEnum	The line ending used for CSV job data, marking the end of a data row. The default is `LF`. Valid values are: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character	Optional
`object`	string	The object type for the data being processed. Use only a single object type per job.	Required

Property	Type	Description	Required or Optional
`operation`	OperationEnum	The processing operation for the job. Valid values are: • `insert` • `delete` • `update` • `upsert`	Required

For multipart requests, the request body can also include CSV record data. See Usage Notes for more details.

Response Body

Property	Type	Description
`apiVersion`	string	The API version that the job was created in.
`columnDelimiter`	ColumnDelimiterEnum	The column delimiter used for CSV job data. Values include: • `BACKQUOTE`—backquote character (`) • `CARET`—caret character (^) • `COMMA`—comma character (,) which is the default delimiter • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character
`concurrencyMode`	ConcurrencyModeEnum	The concurrency mode for the job. Values include: • `Parallel`: Process records in parallel mode. This is the default value. • `Serial`: Process records in serial mode. Processing in parallel can cause database contention. When this is severe, the job can fail. If you're experiencing this issue, submit the job with `serial` concurrency mode. This mode guarantees that records are processed serially, but can significantly increase the processing time.
`contentType`	ContentType	The format of the data being processed. Only CSV is supported.
`contentUrl`	URL	The URL to use for Upload Job Data requests for this job. Only valid if the job is in `Open` state.
`createdById`	string	The ID of the user who created the job.
`createdDate`	dateTime	The date and time in the UTC time zone when the job was created.
`externalIdFieldName`	string	The name of the external ID field for an upsert.
`id`	string	Unique ID for this job.
`jobType`	JobTypeEnum	The job's type. Values include: • `BigObjectIngest`—BigObjects job

Property	Type	Description
		• `Classic`—Bulk API 1.0 job • `V2Ingest`—Bulk API 2.0 job
`lineEnding`	LineEndingEnum	The line ending used for CSV job data. Values include: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character
`object`	string	The object type for the data being processed.
`operation`		The processing operation for the job. Values include: • `insert` • `delete` • `update` • `upsert`
`state`	JobStateEnum	The current state of processing for the job. Values include: • `Open`—The job has been created, and data can be added to the job. • `UploadComplete`—No new data can be added to this job. You can't edit or save a closed job. • `Aborted`—The job has been aborted. You can abort a job if you created it or if you have the "Manage Data Integrations" permission. • `JobComplete`—The job was processed by Salesforce. • `Failed`—The job has failed. Job data that was successfully processed isn't rolled back.
`systemModstamp`	dateTime	Date and time in the UTC time zone when the job finished.

Usage Notes

For small amounts of job data (20,000 characters or less), you can create a job and upload all the data for a job using a multipart request. The following example request header and body uses a multipart format to contain both job information and job data.

```
Content-Type: multipart/form-data; boundary=BOUNDARY
```

```
--BOUNDARY
Content-Type: application/json
Content-Disposition: form-data; name="job"

{
  "object":"Contact",
  "contentType":"CSV",
  "operation":"insert"
}

--BOUNDARY
Content-Type: text/csv
Content-Disposition: form-data; name="content"; filename="content"
```

```
(Content of your CSV file)
--BOUNDARY--
```

Upload Job Data

Uploads data for a job using CSV data you provide.

URI

/services/data/v*XX.X*/jobs/ingest/*jobID*/batches

This URI is an example. Use the URI provided from Salesforce as described in Usage Notes.

Formats

text/csv

HTTP Method

PUT

Authentication

`Authorization: Bearer token`

Parameters

None.

Request Body

CSV file with record data.

Response Body

None. Returns a status code of 201 (Created), which indicates that the job data was successfully received by Salesforce.

Usage Notes

Use the resource URL provided by Salesforce to upload job data. The resource URL is provided in the `contentUrl` field in the response from Create a Job, or the response from a Job Info request on an open job.

A request can provide CSV data that does not in total exceed 150 MB of base64 encoded content. When job data is uploaded, it is converted to base64. This conversion can increase the data size by approximately 50%. To account for the base64 conversion increase, upload data that does not exceed 100 MB.

Don't delete your local CSV data until you've confirmed that all records were successfully processed by Salesforce. If a job fails, use the successful results, failed results, and unprocessed records resources to determine what records from your CSV data you need to resubmit.

Close or Abort a Job

Closes or aborts a job. If you close a job, Salesforce queues the job and uploaded data for processing, and you can't add any additional job data. If you abort a job, the job does not get queued or processed.

URI

/services/data/v*XX.X*/jobs/ingest/*jobID*

Formats

JSON

HTTP Method

PATCH

Authentication

`Authorization: Bearer token`

Parameters

None.

Request Body

Property	Type	Description	Required or Optional
`state`	JobStateEnum	The state to update the job to. Use `UploadComplete` to close a job, or `Aborted` to abort a job.	Required

Response Body

Property	Type	Description
`apiVersion`	string	The API version that the job was created in.
`columnDelimiter`	ColumnDelimiterEnum	The column delimiter used for CSV job data. Values include: • `BACKQUOTE`—backquote character (`) • `CARET`—caret character (^) • `COMMA`—comma character (,) which is the default delimiter • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character
`concurrencyMode`	ConcurrencyModeEnum	The concurrency mode for the job. Values include: • `Parallel`: Process records in parallel mode. This is the default value. • `Serial`: Process records in serial mode. Processing in parallel can cause database contention. When this is severe, the job can fail. If you're experiencing this issue, submit the job with `serial` concurrency mode. This mode guarantees that records are processed serially, but can significantly increase the processing time.
`contentType`	ContentType	The format of the data being processed. Only CSV is supported.
`contentUrl`	URL	The URL to use for Upload Job Data requests for this job. Only valid if the job is in `Open` state.
`createdById`	string	The ID of the user who created the job.
`createdDate`	dateTime	The date and time in the UTC time zone when the job was created.
`externalIdFieldName`	string	The name of the external ID field for an upsert.
`id`	string	Unique ID for this job.
`jobType`	JobTypeEnum	The job's type. Values include: • `BigObjectIngest`—BigObjects job

Property	Type	Description
		• `Classic`—Bulk API 1.0 job • `V2Ingest`—Bulk API 2.0 job
`lineEnding`	LineEndingEnum	The line ending used for CSV job data. Values include: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character
`object`	string	The object type for the data being processed.
`operation`		The processing operation for the job. Values include: • `insert` • `delete` • `update` • `upsert`
`state`	JobStateEnum	The current state of processing for the job. Values include: • `Open`—The job has been created, and data can be added to the job. • `UploadComplete`—No new data can be added to this job. You can't edit or save a closed job. • `Aborted`—The job has been aborted. You can abort a job if you created it or if you have the "Manage Data Integrations" permission. • `JobComplete`—The job was processed by Salesforce. • `Failed`—The job has failed. Job data that was successfully processed isn't rolled back.
`systemModstamp`	dateTime	Date and time in the UTC time zone when the job finished.

Delete a Job

Deletes a job. To be deleted, a job must have a state of `UploadComplete`, `JobComplete`, `Aborted`, or `Failed`.

URI

/services/data/v*`XX.X`*/jobs/ingest/*`jobID`*

Formats

JSON

HTTP Method

DELETE

Authentication

`Authorization: Bearer` ***`token`***

Parameters

None.

Request Body
None required.

Response Body
None. Returns a status code of 204 (No Content), which indicates that the job was successfully deleted.

Usage Notes
When a job is deleted, job data stored by Salesforce is also deleted and job metadata information is removed. The job will no longer display in the Bulk Data Load Jobs page in Salesforce.

Get All Jobs

Retrieves all jobs in the org.

URI
/services/data/v*XX.X*/jobs/ingest

Formats
JSON

HTTP Method
GET

Authentication
`Authorization: Bearer` ***`token`***

Parameters

Parameter	Description
`concurrencyMode`	Filters jobs based on concurrency mode. Valid values include: • `Parallel`: Process records in parallel mode. This is the default value. • `Serial`: Process records in serial mode. Processing in parallel can cause database contention. When this is severe, the job can fail. If you're experiencing this issue, submit the job with `serial` concurrency mode. This mode guarantees that records are processed serially, but can significantly increase the processing time.
`isPkChunkingEnabled`	If set to `true`, filters jobs with PK chunking enabled.
`jobType`	Filters jobs based on job type. Valid values include: • `BigObjectIngest`—BigObjects job • `Classic`—Bulk API 1.0 job • `V2Ingest`—Bulk API 2.0 job
`queryLocator`	Use `queryLocator` with a locator value to get a specific set of job results. Get All Jobs returns up to 1000 result rows per request, along with a `nextRecordsUrl` value that contains the locator value used to get the next set of results.

Request Body
None required.

Response Body

Property	Type	Description
`done`	boolean	Indicates whether there are more jobs to get. If `false`, use the `nextRecordsUrl` value to retrieve the next group of jobs.
`records`	JobInfo[ ]	Contains information for each retrieved job.
`nextRecordsUrl`	URL	A URL that contains a query locator used to get the next set of results in a subsequent request if `done` isn't `true`.

JobInfo

Property	Type	Description
`apiVersion`	string	The API version that the job was created in.
`columnDelimiter`	ColumnDelimiterEnum	The column delimiter used for CSV job data. Values include: • `BACKQUOTE`—backquote character (`) • `CARET`—caret character (^) • `COMMA`—comma character (,) which is the default delimiter • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character
`concurrencyMode`	ConcurrencyModeEnum	The concurrency mode for the job. Values include: • `Parallel`: Process records in parallel mode. This is the default value. • `Serial`: Process records in serial mode. Processing in parallel can cause database contention. When this is severe, the job can fail. If you're experiencing this issue, submit the job with `serial` concurrency mode. This mode guarantees that records are processed serially, but can significantly increase the processing time.
`contentType`	ContentType	The format of the data being processed. Only CSV is supported.
`contentUrl`	URL	The URL to use for Upload Job Data requests for this job. Only valid if the job is in `Open` state.
`createdById`	string	The ID of the user who created the job. Create the batch with the same user.
`createdDate`	dateTime	The date and time in the UTC time zone when the job was created.
`id`	string	Unique ID for this job.
`jobType`	JobTypeEnum	The job's type. Values include: • `BigObjectIngest`: BigObjects job

Property	Type	Description
		• `Classic`: Bulk API 1.0 job • `V2Ingest`: Bulk API 2.0 job
`lineEnding`	LineEndingEnum	The line ending used for CSV job data. Values include: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character
`object`	string	The object type for the data being processed.
`operation`		The processing operation for the job. Values include: • `insert` • `delete` • `update` • `upsert`
`state`	JobStateEnum	The current state of processing for the job. Values include: • `Open`—The job has been created, and data can be added to the job. • `UploadComplete`—No new data can be added to this job. You can't edit or save a closed job. • `Aborted`—The job has been aborted. You can abort a job if you created it or if you have the "Manage Data Integrations" permission. • `JobComplete`—The job was processed by Salesforce. • `Failed`—The job has failed. Job data that was successfully processed isn't rolled back.
`systemModstamp`	dateTime	Date and time in the UTC time zone when the job finished.

Get Job Info

Retrieves detailed information about a job.

URI

/services/data/v*XX.X*/jobs/ingest/*jobID*

Formats

JSON

HTTP Method

GET

Authentication

`Authorization: Bearer token`

Parameters

None.

Request Body

None required.

Response Body

Property	Type	Description
`apexProcessingTime`	long	The number of milliseconds taken to process triggers and other processes related to the job data. This doesn't include the time used for processing asynchronous and batch Apex operations. If there are no triggers, the value is 0.
`apiActiveProcessingTime`	long	The number of milliseconds taken to actively process the job and includes `apexProcessingTime`, but doesn't include the time the job waited in the queue to be processed or the time required for serialization and deserialization.
`apiVersion`	string	The API version that the job was created in.
`columnDelimiter`	ColumnDelimiterEnum	The column delimiter used for CSV job data. Values include: • `BACKQUOTE`—backquote character (`) • `CARET`—caret character (^) • `COMMA`—comma character (,) which is the default delimiter • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character
`concurrencyMode`	ConcurrencyModeEnum	The concurrency mode for the job. Values include: • `Parallel`: Process records in parallel mode. This is the default value. • `Serial`: Process records in serial mode. Processing in parallel can cause database contention. When this is severe, the job can fail. If you're experiencing this issue, submit the job with `serial` concurrency mode. This mode guarantees that records are processed serially, but can significantly increase the processing time.
`contentType`	ContentType	The format of the data being processed. Only CSV is supported.
`contentUrl`	URL	The URL to use for Upload Job Data requests for this job. Only valid if the job is in `Open` state.
`createdById`	string	The ID of the user who created the job.
`createdDate`	dateTime	The date and time in the UTC time zone when the job was created.
`externalIdFieldName`	string	The name of the external ID field for an upsert.
`id`	string	Unique ID for this job.

Property	Type	Description
`jobType`	JobTypeEnum	The job's type. Values include: • `BigObjectIngest`: BigObjects job • `Classic`: Bulk API 1.0 job • `V2Ingest`: Bulk API 2.0 job
`lineEnding`	LineEndingEnum	The line ending used for CSV job data. Values include: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character
`numberRecordsFailed`	int	The number of records that were not processed successfully in this job.
`numberRecordsProcessed`	int	The number of records already processed.
`object`	string	The object type for the data being processed.
`operation`		The processing operation for the job. Values include: • `insert` • `delete` • `update` • `upsert`
`retries`	int	The number of times that Salesforce attempted to save the results of an operation. The repeated attempts are due to a problem, such as a lock contention.
`state`	JobStateEnum	The current state of processing for the job. Values include: • `Open`: The job has been created, and job data can be uploaded to the job. • `UploadComplete`: All data for a job has been uploaded, and the job is ready to be queued and processed. No new data can be added to this job. You can't edit or save a closed job. • `Aborted`: The job has been aborted. You can abort a job if you created it or if you have the "Manage Data Integrations" permission. • `JobComplete`: The job was processed by Salesforce. • `Failed`: The job has failed. Job data that was successfully processed isn't rolled back.
`systemModstamp`	dateTime	Date and time in the UTC time zone when the job finished.
`totalProcessingTime`	long	The number of milliseconds taken to process the job.

Get Job Successful Record Results

Retrieves a list of successfully processed records for a completed job.

URI

/services/data/v*XX.X*/jobs/ingest/*jobID*/successfulResults/

Formats

CSV

HTTP Method

GET

Authentication

`Authorization: Bearer token`

Parameters

None.

Request Body

None required.

Response Body

The response body is a CSV file that all the records that the job successfully processed. Each row corresponds to a successfully processed record and contains the following information.

Property	Type	Description
`sf__Created`	boolean	Indicates if the record was created.
`sf__Id`	string	ID of the record that was successfully processed.
`Fields from the original CSV request data`	*various*	Field data for the row that was provided in the original job data upload request.

Usage Notes

The order of records in the response is not guaranteed to match the ordering of records in the original job data.

Get Job Failed Record Results

Retrieves a list of failed records for a completed job.

URI

/services/data/v*XX.X*/jobs/ingest/*jobID*/failedResults/

Formats

CSV

HTTP Method

GET

Authentication

`Authorization: Bearer token`

Parameters

None.

Request Body

None required.

Response Body

The response body is a CSV file that all the records that encountered an error while being processed by the job. Each row corresponds to a failed record and contains the following information.

Property	Type	Description
`sf__Error`	Error	Error code and message, if applicable.
`sf__Id`	string	ID of the record that had an error during processing, if applicable.
`Fields from the original CSV request data`	*various*	Field data for the row that was provided in the original job data upload request.

Usage Notes

The order of records in the response is not guaranteed to match the ordering of records in the original job data.

Get Job Unprocessed Record Results

Retrieves a list of unprocessed records for a completed job.

URI

/services/data/v*`XX.X`*/jobs/ingest/*`jobID`*/unprocessedrecords/

Formats

CSV

HTTP Method

GET

Authentication

`Authorization: Bearer` ***`token`***

Parameters

None.

Request Body

None required.

Response Body

The response body is a CSV file that contains all the records that were not processed by the job.

A job that is interrupted or otherwise fails to complete can result in rows that aren't processed. Unprocessed rows are not the same as failed rows. Failed rows are processed but encounter an error during processing.

Each row corresponds to a unprocessed record and contains the following information.

Property	Type	Description
`Fields from the original CSV request data`	*various*	Field data for the row that was provided in the original job data upload request.

Usage Notes

The order of records in the response is not guaranteed to match the ordering of records in the original job data.