Bulk API 2.0 and Bulk API Developer Guide

Version 55.0, Summer '22

@salesforcedocs

Last updated: June 7, 2022

CONTENTS

CHAPTER 1 Introduction to Bulk API 2.0

The REST-based Bulk API 2.0 provides a programmatic option to asynchronously upload, query, or delete large data sets in your Salesforce org. Any data operation that includes more than 2,000 records is a good candidate for Bulk API 2.0 to successfully prepare, execute, and manage an *asynchronous* workflow that makes use of the Bulk framework. Jobs with fewer than 2,000 records should involve "bulkified" *synchronous* calls in REST (for example, Composite) or SOAP. Using the API requires basic familiarity with software development, web services, and the Salesforce user interface. This API is enabled by default for Performance, Unlimited, Enterprise, and Developer Editions. The "API Enabled" permission must be enabled.

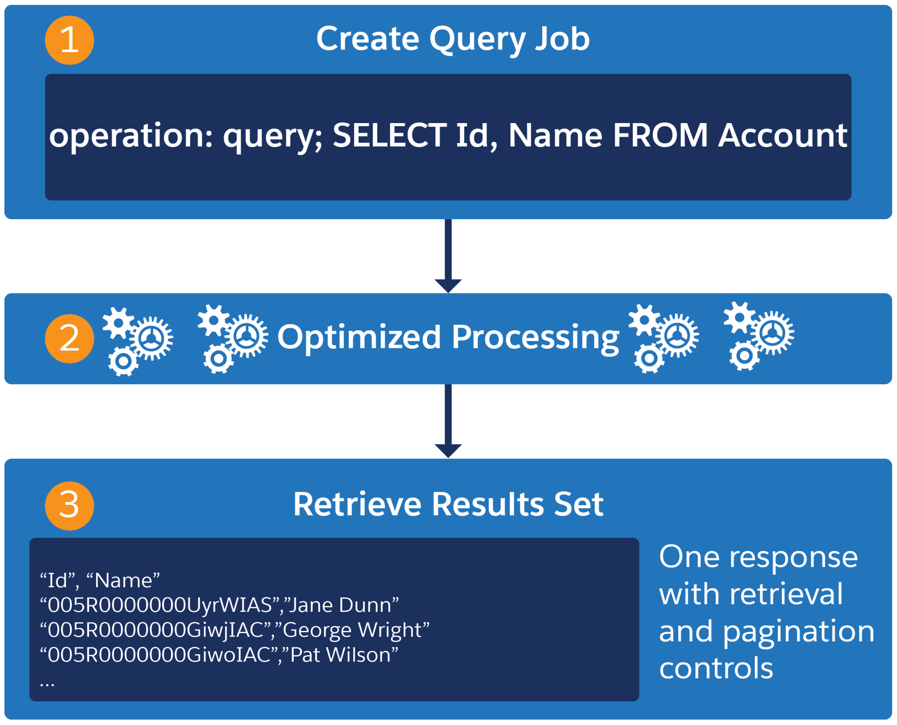
What's the Difference Between Bulk API 2.0 and Bulk API?

Although Bulk API 2.0's predecessor, "Bulk API", is available, use Bulk API 2.0 instead of Bulk API if you want a more streamlined workflow. Bulk API 2.0 provides a simple interface to load large amounts of data into your Salesforce org and to perform bulk queries on your org data. Its design is more consistent and better integrated with other Salesforce APIs. Bulk API 2.0 also has the advantage of future innovation.

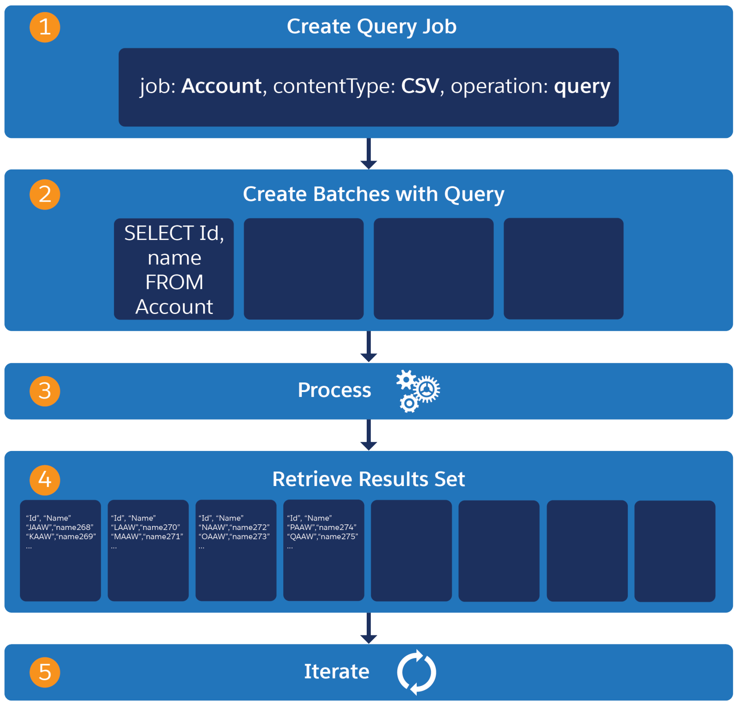
Bulk API 2.0 allows for:

- Less client-side code writing.
- Easy-to-monitor job status.
- Automatic retry of failed records.
- Support for parallel processing.
- Fewer calls required to complete ingest or query workflows.
- Easier batch management.

Here's an example of the Bulk API 2.0 query workflow:


1
Create Query Job
operation: query; SELECT Id, Name FROM Account
2
Optimized Processing
3
Retrieve Results Set
“Id”, “Name”
“005R0000000UyrWIAS”,”Jane Dunn”
“005R0000000GiwjIAC”,”George Wright”
“005R0000000GiwoIAC”,”Pat Wilson”
…
One response with retrieval and pagination controls

Bulk API's query workflow is more complex - requiring the creation of batches and iterating through the retrieval of result sets:


1
Create Query Job
job: Account, contentType: CSV, operation: query
2
Create Batches with Query
SELECT Id, name FROM Account
3
Process
4
Retrieve Results Set
“Id”, “Name” “JAAW”,“name268” “KAAW”,“name269” …
“Id”, “Name” “LAAW”,“name270” “MAAW”,“name271” …
“Id”, “Name” “NAAW”,“name272” “OAAW”,“name273” …
“Id”, “Name” “PAAW”,“name274” “QAAW”,“name275” …
5
Iterate

If the feature set and limits are a unique match to your project requirements, use Bulk API.

This table shows a basic feature set comparison between Bulk API 2.0 and Bulk API.

Feature	Bulk API 2.0	Bulk API
Ingest Availability	41.0 and later	16.0 and later
Query Availability	47.0 and later	21.0 and later
Authentication	Supports all OAuth 2.0 flows supported by other Salesforce REST APIs.	None. Requires a special X-SFDC-Session header fetched with SOAP API's `login()` call.
Ingest Data Format	CSV	CSV, XML, JSON, and binary attachment processing
Large File Batching	Simplifies uploading large amounts of data by breaking the data into batches and providing parallelism automatically. Upload a CSV file with your record data and check back when the results are ready. All results are returned from one endpoint.	Large files must be batched manually, either with custom code or by hand.
Support for big objects	• Ingest • Query	• Ingest • Query
Query Job Optimization	Automatically performs PK chunking.	PK chunking is manually invoked and configured.
Query Results Retrieval	All in a single endpoint.	Iterate through the retrieval of individual result sets.
Daily Upload Limits	Limits by total *records* uploaded per day. Available to clients via the REST API `/limits` endpoint.	Limits by quantity of *batches* per day
Data Loader Compatibility	No	Yes

For a detailed comparison of Bulk API 2.0 and Bulk API limits, see Limits on page 7 in this document.

SEE ALSO:

Bulk API 2.0 Older Documentation

How Requests Are Processed

Bulk ingest jobs allow you to upload records to your org by using a CSV file representation. Bulk query jobs return records based on the specified query. A Bulk API job specifies which object is being processed (for example, Account or Opportunity) and what type of action is being used (insert, upsert, update, or delete). You process a set of records by creating a job that contains one or more batches. Whether you create an ingest or query job, Salesforce automatically optimizes your request to process the job as quickly as possible and to minimize timeouts or other failures.

Job States

When you create job requests with Bulk API 2.0, Salesforce provides a job "state" to describe the progress or outcome of the job. You can manually Check the status of the job, or you can view job state from within the Salesforce UI. From Setup, in the Quick Find box, enter *`Bulk Data Load Jobs`*, and then select **Bulk Data Load Jobs**. The following table summarizes Bulk API 2.0 job states during job creation and processing.

Job Phase	State	Description
Creation	`Open`	An ingest job was created and is open for data uploads.
Creation	`UploadComplete`	(*Ingest*) All job data has been uploaded. The job is queued and ready to be processed. (*Query*) Salesforce has put the query job in the queue.
Processing	`InProgress`	The job is being processed by Salesforce. Operations include automatic, optimized batching or chunking of job data, and processing of job operations.
Outcome	`JobComplete`	The job was processed.
Outcome	`Failed`	The job couldn't be processed successfully.
Outcome	`Aborted`	The job was canceled by the job creator, or by a user with the "Manage Data Integrations" permission.

Ingest Jobs

While processing ingest jobs, Salesforce Bulk API 2.0 automatically divides your job's data into multiple batches to improve performance.

Salesforce creates a separate batch for every 10,000 records in your job data, up to a daily maximum of 150,000,000 records. If the limit is exceeded while processing your job data, the remaining data isn't processed. The ingest job is marked as having failed.

Just as a job can fail, so can an individual batch. If Salesforce can't process all the records in a batch within 10 minutes, the batch fails. Salesforce automatically retries failed batches up to a maximum of 10 times. If the batch still can't be processed after 10 retries, the entire ingest job is moved to the `Failed` state and remaining job data isn't processed.

If there's a failure, create a new ingest job to process the records that weren't processed.

To determine what records weren't processed and what errors occurred, use the Failed Record Results and Unprocessed Record Results resources.

Query Jobs

Bulk API 2.0 query jobs enable asynchronous processing of SOQL queries. Instead of manually configuring batches, Bulk API 2.0 query jobs automatically determine the best way to divide your query job into smaller chunks, helping to avoid failures or timeouts. The API automatically handles retries. If you receive a message that the API retried more than 15 times, apply a filter criteria and try again. When you get the results of a query job, the response body is always compressed.

Bulk API 2.0 is optimized to chunk large query jobs with the following objects:

- Account
- AccountContactRelation
- AccountTeamMember
- AiVisitSummary
- Asset
- AssignedResource
- B2BMktActivity
- B2BMktProspect
- Campaign
- CampaignMember
- CandidateAnswer
- Case
- CaseArticle
- CaseComment
- ChangeRequest
- Claim
- ClaimParticipant
- Contact
- ContentDistribution
- ContentDocument
- ContentVersion
- ContractLineItem
- ConversationDefinitionEventLog
- ConversationEntry
- ConversationReason
- ConversationReasonExcerpt
- ConversationReasonGroup
- CustomerProperty
- EinsteinAnswerFeedback
- EmailMessage
- EngagementScore
- Event
- EventRelation
- FeedItem
- Incident
- Individual
- InsurancePolicy
- InsurancePolicyAsset
- InsurancePolicyParticipant

- Lead
- LeadInsight
- LinkedArticle
- LiveChatTranscript
- LoginHistory
- LoyaltyAggrPointExprLedger
- LoyaltyLedger
- LoyaltyMemberCurrency
- LoyaltyMemberTier
- LoyaltyPartnerProduct
- LoyaltyProgramMbrPromotion
- LoyaltyProgramMember
- LoyaltyProgramPartner
- LoyaltyProgramPartnerLedger
- MlRetrainingFeedback
- Note
- ObjectTerritory2Association
- Opportunity
- OpportunityContactRole
- OpportunityHistory
- OpportunityLineItem
- OpportunitySplit
- OpportunityTeamMember
- Order
- OrderItem
- Pricebook2
- PricebookEntry
- Problem
- Product2
- ProductConsumed
- ProductRequired
- QuickText Quote
- QuoteLineItem
- ReplyText
- ScoreIntelligence
- ServiceContract Task
- TaskRelation
- TermDocumentFrequency
- TransactionJournal
- User

- UserRole
- VoiceCall
- VoiceCallRecording
- Voucher
- WebCart
- WorkloadUnit
- WorkOrder
- WorkOrderLineItem
- WorkPlan
- WorkPlanTemplate

This optimization also includes custom objects, and any Sharing and History tables that support standard objects.

SEE ALSO:

Bulk API 2.0 Older Documentation

Limits

Set Up and Maintain Your Salesforce Organization: Manage Bulk Data Load Jobs

Limits

Learn about the importance of limits, and compare the limits and allocations of Bulk API 2.0 and Bulk API. For Bulk API 2.0, we simplified limits, which are available to clients via the REST API `/limits` endpoint.

Considering Limits

Limits are in place to ensure optimal performance for all customers and to provide fair access to system resources. Each org is only able to handle a certain number of API requests within a 24-hour period. Budget your overall API consumption to account for what each integration does against the org.

Questions that might help to plan for limits:

- How many other integrations are making API requests into your org?
- How close does your org come to reaching its entitled request limit each day?
- How many API requests per day would be required in order to address your use cases and data volume?
- Of the APIs that could do the job you're planning, what are their limits characteristics?

So consider both what your new implementation is attempting to do as well as what existing integrations are doing to make sure your workloads won't be interrupted.

Batch Allocations

You can submit up to 15,000 batches per rolling 24-hour period. This allocation is shared between Bulk API and Bulk API 2.0, so every batch that is processed in Bulk API or Bulk API 2.0 counts towards this allocation.

In Bulk API 2.0, only ingest jobs consume batches. Query jobs don't. For details, see How Requests Are Processed in the *Bulk API 2.0 Developer Guide*.

In Bulk API 2.0, batches are created for you automatically. In Bulk API, you must create the batches yourself.

General Limits

Item	Bulk API Limit	Bulk API 2.0 Limit
Batch and job lifespan	Batches and jobs that are older than seven days are removed from the queue if batches are in a terminal state (completed, aborted, or failed), regardless of their respective job status. The seven days are measured from the youngest batch associated with a job, or the age of the job if there are no batches. You can't create batches associated with a job that is more than 24 hours old. Batches in a non-terminal state that are older than seven days are periodically cleaned up with their respective jobs.	Jobs in a terminal state (completed, aborted, or failed) that are older than seven days are deleted. Jobs in a non-terminal state that are older than seven days are periodically cleaned up.
Binary content	• The length of any file name can't exceed 512 bytes. • A zip file can't exceed 10 MB. • The total size of the unzipped content can't exceed 20 MB. • A maximum of 1,000 files can be contained in a zip file. Directories don't count toward this total.	N/A
Maximum time that a job can remain open	24 hours	The same. (But this only applies to ingest jobs, not query jobs.)

Limits Specific to Ingest Jobs

Item	Bulk API Limit	Bulk API 2.0 Limit
Maximum number of records uploaded per 24-hour rolling period	150,000,000 (15,000 batches x 10,000 records per batch maximum)	150,000,000
Batch processing time	Batches are processed in chunks. The chunk size depends on the API version. In API version 20.0 and earlier, the chunk size is 100 records. In API version 21.0 and later, the chunk size is 200 records. If it takes longer than 10 minutes to process a whole batch, the Bulk API places the remainder of the batch back in the queue for later processing. If the Bulk API continues to exceed the 10-minute limit on subsequent attempts, the batch is placed back in the queue and reprocessed up to 10 times before the batch is permanently marked as failed.	Same as Bulk API
Maximum time before a batch is retried	10 minutes	The API automatically handles retries. If you receive a message that the API retried more than 10 times, use a smaller upload file and try again.
Maximum file size	10 MB	150 MB

Item	Bulk API Limit	Bulk API 2.0 Limit
Maximum number of characters in a field	131072	Same as Bulk API
Maximum number of fields in a record	5,000	Same as Bulk API
Maximum number of characters in a record	400,000	Same as Bulk API
Maximum number of records in a batch	10,000	N/A
Maximum number of characters for all the data in a batch	10,000,000	N/A

Limits Specific to Query Jobs

Item	Bulk API Limit	Bulk API 2.0 Limit
Number of attempts to query	15 attempts at 10 minutes each to process the batch. There's also a 2-minute limit on the time to process the query. If more than 15 attempts are made for the query, an error message of "Tried more than fifteen times" is returned. If the query takes more than 2 minutes to process, a QUERY_TIMEOUT error is returned.	The API automatically handles retries. If you receive a message that the API retried more than 15 times, apply a filter criteria and try again.
Number of retrieved files	15 files. If the query returns more than 15 files, add filters to the query to return less data. Bulk batch sizes aren't used for bulk queries.	N/A
Timeout for retrieving query results	20 minutes	Same as Bulk API
Results lifespan	You can retrieve the query job's results within 7 days of job completion.	Same as Bulk API
Maximum retrieved file size	1 GB. If processing of the batch results in 1 GB of retrieved data, then those results are saved to disk, and then the batch is put back on the queue to be resumed later. This also counts as one of the 15 retries.	Same as Bulk API. Additionally, the API client can navigate through the full set of results by using the `locator` and `maxRecords` query parameters. The client isn't bound to a set of files.
Number of query jobs that can be submitted per 24-hour rolling window	See Batch Allocations.	10,000 The current number can be seen in the `DailyBulkV2QueryJobs` value in the

Item	Bulk API Limit	Bulk API 2.0 Limit
		response to the `/vXX.X/limits/` REST API method.
Total query results that can be generated per 24 hour rolling window	N/A	1 TB. The current size can be seen in the `DailyBulkV2QueryFileStorageMB` value in the response to the `/vXX.X/limits/` REST API method.

Number of characters in a SELECT clause.
 	
This limit pertains to all the fields listed between SELECT and FROM. This includes the commas that separate the fields and the quotation marks that Salesforce automatically wraps around each field. Salesforce automatically removes any spaces between fields.
 	
For example, in this query
 	
`SELECT CloseDate,Name, StageName,Amount FROM Opportunity`
 	
Salesforce converts the SELECT clause to
 	
`"CloseDate","Name", "StageName","Amount"`
 	
The number of characters in this example is 39.
	N/A	32,000

Per-Transaction Apex Limits

For Bulk API and Bulk API 2.0 transactions, the effective limit is the higher of the synchronous and asynchronous limits. Limits are detailed in Per-Transaction Apex Limits in *Apex Developer Guide*. For example, the maximum number of Bulk Apex jobs added to the queue with `System.enqueueJob` is the synchronous limit (50), which is higher than the asynchronous limit (1).

Maximum CPU Time Limit

Bulk API and Bulk API 2.0 processes consume a *unique* governor limit for CPU time on Salesforce Servers, isolated from the generic per-transaction Apex limit for maximum CPU time.

Description	Value
Maximum CPU time on the Salesforce servers for Bulk API and Bulk API 2.0	60,000 milliseconds

SEE ALSO:

Bulk API 2.0 Older Documentation

CHAPTER 2 Quick Start: Bulk API 2.0

In this chapter ...

Get up and running with Bulk API 2.0 by sending a few requests to Salesforce. This Quick Start takes you from setting up a basic environment to inserting, upserting, and querying records using Bulk API 2.0. Experience how to use Bulk API 2.0 via cURL in a free Salesforce Developer Edition org by authenticating and following the examples.

Using cURL

Get to know the formatting used with cURL to place calls to Salesforce orgs. This Quick Start uses cURL examples to issue Bulk API 2.0 calls, but you can use any tool or development environment that can make REST requests.

Familiarize yourself with cURL enough to be able to understand the examples in this guide and translate them into the tool that you're using. You'll be attaching files containing the body of the request and must properly format the access token. Use these tips to help you use cURL while working through the Bulk 2.0 Quick Start. For more information about cURL, see the documentation at curl.se.

Attach Request Bodies

Many examples include request bodies—JSON or XML files that contain data for the request. When using cURL, save these files to your local system and attach them to the request using the `—data-binary` or `-d` option.

This example attaches the `new-account.json` file.

```
curl https://MyDomainName.my.salesforce.com/services/data/v55.0/sobjects/Account/ -H
'Authorization Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: application/json" —d @new-account.json -X POST
```

Handle Exclamation Marks in Access Tokens

When you run cURL examples, you can get an error on macOS and Linux systems due to the presence of the exclamation mark (!) in OAuth access tokens. To avoid getting this error, either escape the exclamation mark or use single quotes. To escape the exclamation mark in the access token, insert a backslash before it when the access token is enclosed within double quotes.

```
\!
```

For example, the access token string in this cURL command has the exclamation mark (!) escaped.

```
curl https://MyDomainName.my.salesforce.com/services/data/v55.0/ -H "Authorization: Bearer

00DE0X0A0M0PeLE\!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE"
```

Or, you can enclose the access token within single quotes to not escape the exclamation mark.

```
curl https://MyDomainName.my.salesforce.com/services/data/v55.0/ -H 'Authorization: Bearer

00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
```

Important: All quotes, whether single or double, must be straight quotes, not curly quotes.

Step 1: Set Up a Salesforce Developer Edition Org

This Quick Start suggests using a Developer Edition org. Sign up for a Salesforce Developer Edition org before trying Bulk API 2.0 with this Quick Start.

If you're not already a member of the developer community, go to `developer.salesforce.com/signup`, and follow the instructions for signing up for a Developer Edition account.

You can also use a scratch org or sandbox to follow along with these examples.

Note: Developer Edition orgs have a data storage maximum of 5 MB. This limit doesn't prevent you from working with these examples.

Step 2: Authentication

The first action in an API-based integration is authenticating requests with your Salesforce org. Bulk API 2.0 and Bulk API use different authentication methods.

Bulk API 2.0 is a REST-based API that supports all OAuth 2.0 flows supported by other Salesforce REST APIs. Bulk API 2.0 requires an access token (also known as a "bearer token") for authentication. This topic, and the remainder of this Quick Start, describe getting an access token and using it to make Bulk API 2.0 requests with cURL.

In contrast, Bulk API uses a session ID obtained with an X-SFDC-Session header fetched with SOAP API's `login()` call. For an example, see Step 1: Log In Using the SOAP API on page 93.

Note: These examples use an access token. Any API call that requires a session ID doesn't work with these instructions.

While it's possible to create and authenticate against your own connected app, Salesforce CLI is used in these Quick Start examples for convenience. Effectively, Salesforce CLI is a connected app with which you can authenticate and requires no work to configure.

The examples in this Quick Start use the cURL tool to send HTTP requests that access, create, and manipulate resources in Salesforce. If you use a different tool to send requests, you can use the same elements from the cURL examples to send requests. Although these instructions describe a scenario with a Developer org, they work in the same way with any type of Salesforce org. The cURL tool is pre-installed on many Linux and macOS systems. Windows users can download a version at curl.se. When using HTTPS on Windows, ensure that your system meets the cURL requirements for SSL.

Note: cURL is an open-source tool and isn't supported by Salesforce.

Get an Access Token with Salesforce CLI

Use the access token (also known as a "bearer token") that you get from Salesforce CLI to authenticate cURL requests.

1. Install or update Salesforce CLI. .
   a. If you already have Salesforce CLI installed, update it using the instructions in Update Salesforce CLI.
   b. If you need to Install Salesforce CLI, install the latest version for your operating system.
   c. Verify Your Installation.
2. Log in to your Developer org with Salesforce CLI.
   ```
   sfdx auth:web:login
   ```
   A browser opens to https://login.salesforce.com.
3. In the browser, log in to your Developer org with your user's credentials.
4. In the browser, click **Allow** to allow access.
   At the command line, you see a similar confirmation message.
   ```
   Successfully authorized juliet.capulet@empathetic-wolf-g5qddtr.com with org ID
   00D5fORGIDEXAMPLE
   ```
5. At the command line, get the access token by viewing authentication information about your org.
   ```
   sfdx force:org:display --targetusername <username>
   ```
   For example:
   ```
   sfdx force:org:display --targetusername juliet.capulet@empathetic-wolf-g5qddtr.com
   ```

Example command output:

```
=== Org Description
KEY                VALUE
────────────────
──────────────────────────────────────────────────────────────────────────────────────────
Access Token
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE
Client Id          PlatformCLI
Connected Status   Connected
Id                 00D5fORGIDEXAMPLE
Instance Url       https://MyDomainName.my.salesforce.com
Username           juliet.capulet@empathetic-wolf-g5qddtr.com
```

In the command output, make note of the long Access Token string and the Instance Url string. You need both to make cURL requests.

Note: To get a new token after your access token expires, repeat this step of viewing your authentication information.

Optional Salesforce CLI Shortcuts

After you've authenticated successfully with the SFDX command line, try out these optional shortcuts in your cURL workflow to streamline future authentication with the SDFX CLI.

List My Orgs

```
sfdx force:org:list
```

Lists all the orgs that you've created or authenticated to.

Open My Org

```
sfdx force:org:open -u <user>
```

Opens the specified org (identified by user) in your browser. Because you've successfully authenticated with this org previously using SFDX, it's not required to provide your credentials again.

Display the Access Token for My Org

```
sfdx force:org:display -u <user>
```

Output includes your access token, client ID, connected status, org ID, instance URL, username, and alias, if applicable.

Set an Alias for My Username

For convenience, create an alias for your username so that you don't have to enter the entire Salesforce string. For example, instead of

```
juliet.capulet@empathetic-wolf-g5qddtr.com
```

Create an alias like

```
dev
```

To set the alias in this example, you invoke

```
sfdx alias:set dev=juliet.capulet@empathetic-wolf-g5qddtr.com
```

Use These Commands in a Script

Use the CLI's JSON output by invoking the `--json` flag. Requesting JSON output provides a consistent output format, which is ideal for running scripts. Without the `--json` flag, the CLI can change the output format.

See Also

- Salesforce CLI Setup Guide
- Salesforce CLI Command Reference

Step 3: Bulk Insert

This Bulk API 2.0 example guides you through creating a job to insert new records, uploading data for the job, checking the status, and retrieving the results.

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

1. Copy this CSV formatted list of accounts into a file named `bulkinsert.csv`. You use this file to upload data after creating the job.

   Note: Save all files in this example in your terminal's current working directory.

   The first row of the CSV file lists the field names for the object that you're working with. Each subsequent row corresponds to a record that you want to insert.

```
Name,ShippingCity,NumberOfEmployees,AnnualRevenue,Website,Description
Lorem Ipsum,Milano,2676,912260031,https://ft.com/lacus/at.jsp,"Lorem ipsum dolor sit
amet"
Posuere
Inc,Bodø,141603,896852810,http://webs.com/in/faucibus/orci/luctus/et/ultrices/posuere.json,"consectetur
 adipiscing elit"
Angeles Urban,Aykol,197724,257060529,http://odnoklassniki.ru/sapien.aspx,"sed do eiusmod
 tempor incididunt ut labore et dolore magna aliqua"
Madaline Neubert
Shoes,Xukou,190305,71664061,https://blogs.com/faucibus/orci/luctus/et/ultrices/posuere/cubilia.json,"Ut
 enim ad minim veniam"
Times Online UK,Varadero,121802,58284123,http://timesonline.co.uk/eu/magna.html,"quis
nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat"
The Washington
Post,Hengdaohezi,190944,164329406,http://washingtonpost.com/vestibulum/proin/eu/mi/nulla/ac/enim.png,"Duis
 aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
 pariatur"
Amazon,Quintães,80285,684173825,http://amazon.co.uk/potenti/cras/in/purus/eu.png,"Excepteur
 sint occaecat cupidatat non proident"
```

2. Create a job.

   a. Create a file named `newinsertjob.json`.

   b. Copy this content into the file.

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

URI

```
/services/data/v55.0/jobs/ingest/
```

Example for creating a bulk insert job

```
curl https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/ -H
'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: application/json" -H "Accept: application/json" -H "X-PrettyPrint:1"
 -d @newinsertjob.json -X POST
```

Example response body

The response includes the job `id`, with a job `state` of `Open`.

```
{ "id" : "7505fEXAMPLE4C2AAM",
"operation" : "insert",
"object" : "Account",
"createdById" : "0055fEXAMPLEtG4AAM",
"createdDate" : "2022-01-02T21:33:43.000+0000",
"systemModstamp" : "2022-01-02T21:33:43.000+0000",
"state" : "Open",
"concurrencyMode" : "Parallel",
"contentType" : "CSV",
"apiVersion" : 55.0,
"contentUrl" : "services/data/55.0/jobs/ingest/7505fEXAMPLE4C2AAM/batches",
"lineEnding" : "LF", "columnDelimiter" : "COMMA" }
```

You use the job `id` from this response in the next steps. You can also use the URI in the `contentUrl` field in the next step when you upload your data.

3. Upload your CSV data using the URI in the `contentUrl` field of the response.

   You can upload up to 150 MB per job (after base64 encoding).

   The URI is similar to

   ```
   /services/data/v55.0/jobs/ingest/jobId/batches/
   ```

   Example for uploading data

   ```
   curl
   https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7505fEXAMPLE4C2AAM/batches/
    -H 'Authorization: Bearer
   00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
    -H "Content-Type: text/csv" -H "Accept: application/json" -H "X-PrettyPrint:1"
   --data-binary @bulkinsert.csv -X PUT
   ```

   Example response body

   No response body.

4. Close the job.

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

   URI

   ```
   /services/data/v55.0/jobs/ingest/jobId/
   ```

Example of closing the job

```
curl
https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7505fEXAMPLE4C2AAM/
 -H 'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: application/json; charset=UTF-8" -H "Accept: application/json" -H
"X-PrettyPrint:1" --data-raw '{ "state" : "UploadComplete" }' -X PATCH
```

Example response body

```
{ "id" : "7505fEXAMPLE4C2AAM",
"operation" : "insert",
"object" : "Account",
"createdById" : "0055fEXAMPLEtG4AAM",
"createdDate" : "2022-01-02T21:33:43.000+0000",
"systemModstamp" : "2022-01-02T21:33:43.000+0000",
"state" : "UploadComplete",
"concurrencyMode" : "Parallel",
"contentType" : "CSV",
"apiVersion" : 55.0 }
```

5. Check the job status and results.

URI

```
/services/data/v55.0/jobs/ingest/jobId/
```

Example of checking job status

```
curl
https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7505fEXAMPLE4C2AAM/
 -H 'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Accept: application/json" -H "X-PrettyPrint:1" -X GET
```

Example response body

```
{ "id" : "7505fEXAMPLE4C2AAM",
"operation" : "insert",
"object" : "Account",
"createdById" : "0055fEXAMPLEtG4AAM",
"createdDate" : "2022-01-02T21:33:43.000+0000",
"systemModstamp" : "2022-01-02T21:38:31.000+0000",
"state" : "JobComplete",
"concurrencyMode" : "Parallel",
"contentType" : "CSV",
 "apiVersion" : 55.0,
"jobType" : "V2Ingest",
"lineEnding" : "LF",
"columnDelimiter" : "COMMA",
"numberRecordsProcessed" : 7,
"numberRecordsFailed" : 0,
"retries" : 0,
"totalProcessingTime" : 886,
"apiActiveProcessingTime" : 813,
"apexProcessingTime" : 619 }
```

6. Get successful results.

   After a job is in the `JobComplete` or `Failed` state, you can get details about which records were successfully processed.

   URI

   ```
   /services/data/v55.0/jobs/ingest/jobId/successfulResults/
   ```

   Example of getting successful results

   ```
   curl
   https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7505fEXAMPLE4C2AAM/successfulResults/
    -H 'Authorization: Bearer
   00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
    -H "Content-Type: application/json" -H "Accept: text/csv" -H "X-PrettyPrint:1" -X GET
   ```

   The response contains CSV formatted data, with each row containing a record ID (`sf__Id`) and information on whether that record was successfully processed or not (`sf__Created`).

   Example response body

   ```
   "sf__Id","sf__Created",Name,ShippingCity,NumberOfEmployees,AnnualRevenue,Website,Description
   "0018c00002FInboAAD","true","Lorem
   Ipsum","Milano","2676","9.12260031E8","https://ft.com/lacus/at.jsp","Lorem ipsum dolor
    sit amet"
   "0018c00002FInbpAAD","true","Posuere
   Inc","Bodø","141603","8.9685281E8","http://webs.com/in/faucibus/orci/luctus/et/ultrices/posuere.json","consectetur
    adipiscing elit"
   "0018c00002FInbqAAD","true","Angeles
   Urban","Aykol","197724","2.57060529E8","http://odnoklassniki.ru/sapien.aspx","sed do
   eiusmod tempor incididunt ut labore et dolore magna aliqua"
   "0018c00002FInbrAAD","true","Madaline Neubert
   Shoes","Xukou","190305","7.1664061E7","https://blogs.com/faucibus/orci/luctus/et/ultrices/posuere/cubilia.json","Ut
    enim ad minim veniam"
   "0018c00002FInbsAAD","true","Times Online
   UK","Varadero","121802","5.8284123E7","http://timesonline.co.uk/eu/magna.html","quis
   nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat"
   "0018c00002FInbtAAD","true","The Washington
   Post","Hengdaohezi","190944","1.64329406E8","http://washingtonpost.com/vestibulum/proin/eu/mi/nulla/ac/enim.png","Duis
    aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
    pariatur"
   "0018c00002FInbuAAD","true","Amazon","Quintães","80285","6.84173825E8","http://amazon.co.uk/potenti/cras/in/purus/eu.png","Excepteur
    sint occaecat cupidatat non proident"
   ```

   To get details about records that encountered an error during processing, use a GET request using the `failedResults` resource. To make sure that you're looking at the complete result set, look at unprocessed records by using the `unprocessedRecords` resource. See Get Job Unprocessed Record Results.

Step 4: Bulk Insert with a Multipart Request

This Bulk API 2.0 example guides you through creating a job to insert new records, uploading data for the job, checking status, and retrieving the results. This example uses a single, multipart request to create the job and upload the data.

In this example, `BOUNDARY` is used at the start, middle, and end of the file to mark the request body boundaries between the job's details and tits CSV data.

1. Copy this multipart, JSON-formatted content into a file named `newmultipartjob.json`.

   Note: Save all files in this example in your terminal's current working directory.

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

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

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

FirstName,LastName,MailingCity
Astro,Nomical,San Francisco
Hootie,McOwl,San Francisco
Appy,Camper,San Francisco
Earnie,Badger,San Francisco
--BOUNDARY--
```

2. Create a job.

   URI

```
/services/data/v55.0/jobs/ingest/
```

   Example of creating a job

```
curl https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/ -H
'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: multipart/form-data; boundary=\"BOUNDARY\"" -H "Accept:
application/json" -H "X-PrettyPrint:1" --data-binary @newmultipartjob.json -X POST
```

   The response includes the job `id`, with a job `state` of `UploadComplete`. You use the job `id` in the next steps.

   Example response body

```
{
  "id" : "7303gEXAMPLE4X2QAN",
  "operation" : "insert",
  "object" : "Contact",
  "createdById" : "0055fEXAMPLEtG4AAM",
  "createdDate" : "2022-01-02T19:26:52.000+0000",
  "systemModstamp" : "2022-01-02T19:26:52.000+0000",
  "state" : "UploadComplete",
  "concurrencyMode" : "Parallel",
  "contentType" : "CSV",
  "apiVersion" : 55.0,
  "lineEnding" : "LF",
  "columnDelimiter" : "COMMA"
}
```

After you create a multipart job, it's closed automatically. You don't need to manually close the job.

3. Check the job status and results with this URI.

   URI

   ```
   /services/data/v55.0/jobs/ingest/jobId/
   ```

   Example of checking job status and results

   ```
   curl
   https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7303gEXAMPLE4X2QAN/
    -H 'Authorization: Bearer
   00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
    -H "Accept: application/json" -H "X-PrettyPrint:1" -X GET
   ```

   Example response body

   ```
   {
     "id" : "7303gEXAMPLE4X2QAN",
     "operation" : "insert",
     "object" : "Contact",
     "createdById" : "0055fEXAMPLEtG4AAM",
     "createdDate" : "2022-01-02T19:54:04.000+0000",
     "systemModstamp" : "2022-01-02T19:54:05.000+0000",
     "state" : "JobComplete",
     "concurrencyMode" : "Parallel",
     "contentType" : "CSV",
     "apiVersion" : 55.0,
     "jobType" : "V2Ingest",
     "lineEnding" : "LF",
     "columnDelimiter" : "COMMA",
     "numberRecordsProcessed" : 4,
     "numberRecordsFailed" : 0,
     "retries" : 0,
     "totalProcessingTime" : 50,
     "apiActiveProcessingTime" : 6,
     "apexProcessingTime" : 0
   }
   ```

4. Get successful results.

   After a job is in the `JobComplete` or `Failed` state, you can get details about which records were successfully processed.

   URI

   ```
   /services/data/v55.0/jobs/ingest/jobId/successfulResults/
   ```

   Example of getting successful results

   ```
   curl
   https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7303gEXAMPLE4X2QAN/successfulResults/
    -H 'Authorization: Bearer
   00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
    -H "Content-Type: application/json" -H "Accept: text/csv" -H "X-PrettyPrint:1" -X GET
   ```

Example response body

```
"sf__Id","sf__Created",FirstName,LastName,MailingCity
"0038c00002hMS4kAAG","true","Astro","Nomical","San Francisco"
"0038c00002hMS4lAAG","true","Hootie","McOwl","San Francisco"
"0038c00002hMS4mAAG","true","Appy","Camper","San Francisco"
"0038c00002hMS4nAAG","true","Earnie","Badger","San Francisco"
```

To get details about records that encountered an error during processing, use a GET request with the `failedResults` resource. To make sure that you're looking at the complete result set, look at unprocessed records by using the `unprocessedRecords` resource. See Get Job Unprocessed Record Results.

Step 5: Bulk Upsert

This Bulk API 2.0 example guides you through the steps for creating a job to upsert records, uploading data for the job, checking status, and retrieving the results.

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. Bulk API 2.0 uses the external ID field to determine whether a record is used to update an existing record or create a record.

   This example assumes that the external ID field `customExtIdField__c` has been added to the `Account` object.

   To add this custom field in your org with Object Manager, use these properties.

   - Data Type—**text**
   - Field Label—**customExtIdField**
   - Select **External ID**.

   For more information, see Custom Fields in *Salesforce Help*.

2. Create a CSV file containing the records that you want to upsert.

   Note: Save all files in this example in your terminal's current working directory.

   The first row of the CSV file lists the field names for the object that you're working with. Each subsequent row corresponds to a record that you want to insert.

   One column in the CSV file must correspond to the external ID field `customExtIdField__c`.

   For information on preparing CSV files, such as delimiter options and valid date and time formats, see Bulk API 2.0 Ingest on page 29.

   For this example, copy this information into a file named `accountupsert.csv`.

```
customExtIdField__c,name,NumberOfEmployees
123,GenePoint,800
234,"United Oil & Gas, UK",1467
345,"United Oil & Gas, Singapore",348
456,Edge Communications,10045
567,Burlington Textiles Corp of America,5876
678,Dickenson plc,67
789,Grand Hotels & Resorts Ltd,409
890,Express Logistics and Transport,243
901,University of Arizona,9506
1350,United Oil & Gas Corp.,5467
```

```
1579,sForce,40000
2690,University of The Terrific,1257
```

3. Create a job that includes the external ID field.

   Copy this information into a file named `newupsertjob.json`.

```
{
    "object" : "Account",
    "externalIdFieldName" : "customExtIdField__c",
    "contentType" : "CSV",
    "operation" : "upsert",
    "lineEnding" : "LF"
}
```

URI

```
/services/data/v55.0/jobs/ingest/
```

Example for creating a bulk upsert job

```
curl https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/ -H
'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: application/json" -H "X-PrettyPrint:1" -d @newupsertjob.json -X POST
```

Example response body

The response includes the job ID, with a job state of Open. Use the job ID and the URL in the contentUrl field in the next step when you upload your data.

```
{
  "id" : "7476gEXAMPLE4X2ZWO",
  "operation" : "upsert",
  "object" : "Account",
  "createdById" : "0055fEXAMPLEtG4AAM",
  "createdDate" : "2022-01-02T21:57:03.000+0000",
  "systemModstamp" : "2022-01-02T21:57:03.000+0000",
  "state" : "Open",
  "externalIdFieldName" : "customExtIdField__c",
  "concurrencyMode" : "Parallel",
  "contentType" : "CSV",
  "apiVersion" : 55.0,
  "contentUrl" : "services/data/55.0/jobs/ingest/7476gEXAMPLE4X2ZWO/batches",
  "lineEnding" : "LF",
  "columnDelimiter" : "COMMA"
}
```

4. Upload the CSV data file that you created.

URI

For convenience, use the URI in the `contentUrl` field of the response from step 1. The URI is similar to:

```
/services/data/v55.0/jobs/ingest/jobId/batches/
```

Example for uploading data

```
curl
https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7476gEXAMPLE4X2ZWO/batches/
 -H 'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: text/csv" --data-binary @accountupsert.csv -X PUT
```

Example response body

No response body is returned.

5. Close the job.

After you're done submitting data, tell Salesforce servers that the job is ready for processing by closing the job.

Create a JSON file named `upload_complete.json` with the contents:

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

URI

/services/data/*v55.0*/jobs/ingest/*jobId*/

Example of closing the job

```
curl
https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7476gEXAMPLE4X2ZWO/
 -H 'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: application/json" -H "X-PrettyPrint:1" -d @upload_complete.json -X
PATCH
```

Example response body

```
{
  "id" : "7476gEXAMPLE4X2ZWO",
  "operation" : "upsert",
  "object" : "Account",
  "createdById" : "0055fEXAMPLEtG4AAM",
  "createdDate" : "2022-01-02T21:28:22.000+0000",
  "systemModstamp" : "2022-01-02T21:28:22.000+0000",
  "state" : "UploadComplete",
  "externalIdFieldName" : "customExtIdField__c",
  "concurrencyMode" : "Parallel",
  "contentType" : "CSV",
  "apiVersion" : 55.0
}
```

6. Get successful results.

After a job is in the `JobComplete` or `Failed` state, you can get details about which job data records were successfully processed.

URI

```
/services/data/v55.0/jobs/ingest/jobId/successfulResults/
```

Example of getting successful results

```
curl
https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/ingest/7476gEXAMPLE4X2ZWO/successfulResults/
```

```
 -H 'Authorization: Bearer
00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
 -H "Content-Type: application/json" -H "Accept: text/csv" -H "X-PrettyPrint:1" -X GET
```

The response contains CSV formatted data, with each row containing a record ID of successfully processed records.

Example response body

```
"sf__Id","sf__Created",customExtIdField__c,name,NumberOfEmployees
"0018c00002DJIpJAAX","true","123","GenePoint","800"
"0018c00002DJIpKAAX","true","234","United Oil & Gas, UK","1467"
"0018c00002DJIpLAAX","true","345","United Oil & Gas, Singapore","348"
"0018c00002DJIpMAAX","true","456","Edge Communications","10045"
"0018c00002DJIpNAAX","true","567","Burlington Textiles Corp of America","5876"
"0018c00002DJIpOAAX","true","678","Dickenson plc","67"
"0018c00002DJIpPAAX","true","789","Grand Hotels & Resorts Ltd","409"
"0018c00002DJIpQAAX","true","890","Express Logistics and Transport","243"
"0018c00002DJIpRAAX","true","901","University of Arizona","9506"
"0018c00002DJIpSAAX","true","1350","United Oil & Gas Corp.","5467"
"0018c00002DJIpTAAX","true","1579","sForce","40000"
"0018c00002DJIpUAAX","true","2690","University of The Terrific","1257"
```

To get details about records that encountered an error, use the `failedResults` resource. To make sure that you're looking at the complete result set, use the `unprocessedRecords` resource. See Get Job Unprocessed Record Results.

Step 6: Query Jobs

This Bulk API 2.0 example shows you how to create a query job, monitor its progress, and get the job results.

1. Create the job.

   URI

   ```
   /services/data/v55.0/jobs/query
   ```

   Example of creating a bulk query job

   ```
   curl https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/query -H
   'Content-Type: application/json' -H 'Authorization: Bearer
   00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
    -H "X-PrettyPrint:1" --data-raw '{ "operation" : "query", "query" : "SELECT Id, Name
   FROM Account" } ' -X POST
   ```

   The response includes the job `id` and shows the job's state as `UploadComplete`. (You use the job `id` to monitor the job or get its results.)

   Example response body

   ```
   {
     "id" : "7986gEXAMPLE4X2OPT",
     "operation" : "query",
     "object" : "Account",
     "createdById" : "0055fEXAMPLEtG4AAM",
     "createdDate" : "2022-01-02T17:38:59.000+0000",
     "systemModstamp" : "2022-01-02T17:38:59.000+0000",
     "state" : "UploadComplete",
   ```

```
  "concurrencyMode" : "Parallel",
  "contentType" : "CSV",
  "apiVersion" : 55.0,
  "lineEnding" : "LF",
  "columnDelimiter" : "COMMA"
}
```

2. Monitor the job's state using the returned job `id`.

   URI

   ```
   /services/data/v55.0/jobs/query/queryJobId
   ```

   Example of monitoring the state of the query job

   ```
   curl
   https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/query/7986gEXAMPLE4X2OPT
    -H 'Authorization: Bearer
   00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
    -H "X-PrettyPrint:1" -X GET
   ```

   Example response body

   The response shows the current state of the job. Repeat this step until the state is `JobComplete`.

   ```
   {
     "id" : "7986gEXAMPLE4X2OPT",
     "operation" : "query",
     "object" : "Account",
     "createdById" : "0055fEXAMPLEtG4AAM",
     "createdDate" : "2022-01-02T17:38:59.000+0000",
     "systemModstamp" : "2022-01-02T17:39:00.000+0000",
     "state" : "JobComplete",
     "concurrencyMode" : "Parallel",
     "contentType" : "CSV",
     "apiVersion" : 55.0,
     "jobType" : "V2Query",
     "lineEnding" : "LF",
     "columnDelimiter" : "COMMA",
     "numberRecordsProcessed" : 28,
     "retries" : 0,
     "totalProcessingTime" : 153
   }
   ```

3. Get the results of the job.

   URI

   ```
   /services/data/v55.0/jobs/query/queryJobId/results
   ```

   Example of getting results of the job

   ```
   curl
   https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/query/7986gEXAMPLE4X2OPT/results/
    -H 'Authorization: Bearer
   00DE0X0A0M0PeLE!AQcAQH0dMHEXAMPLEzmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1EXAMPLEDUkWe6H34r1AAwOR8B8fLEz6nEXAMPLE'
    -H "Content-Type: application/json; charset=UTF-8" -H "Accept: text/csv" -H
   "X-PrettyPrint:1" -X GET
   ```

The response shows the results of the SOQL query when you created the query job.

Example response body

```
"Id","Name"
"0015f00000BCvReAAL","Sample Account for Entitlements"
"0015f00000BFjNuAAL","University of The Terrific"
"0015f00000C6beUAAR","Edge Communications"
"0015f00000C6beVAAR","Burlington Textiles Corp of America"
"0015f00000C6beWAAR","Pyramid Construction Inc."
"0015f00000C6beXAAR","Dickenson plc"
"0015f00000C6beYAAR","Grand Hotels & Resorts Ltd"
"0015f00000C6beZAAR","United Oil & Gas Corp."
"0015f00000C6beaAAB","Express Logistics and Transport"
"0015f00000C6bebAAB","University of Arizona"
"0015f00000C6becAAB","United Oil & Gas, UK"
"0015f00000C6bedAAB","United Oil & Gas, Singapore"
"0015f00000C6beeAAB","GenePoint"
```

This example returns a small result set, and it’s easy to see the complete results. Queries that return larger results spread them across a sequence of result sets. To see the other result sets, use the locator to fetch the next set of results. For more information, see Get Results for a Query Job.

CHAPTER 3 Bulk API 2.0

In this chapter ...

Perform ingest and query operations with Salesforce Bulk API 2.0.

Bulk API 2.0 Ingest

With Bulk API 2.0, you can insert, update, upsert, or delete large data sets. Prepare a comma-separated value (CSV) file representation of the data you want to upload, create a job, upload job data, and let Salesforce take care of the rest within your org.

Use 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 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.
- Each field-name header in the file must be the same as the field's Field Name (for standard fields) or API Name (for custom fields). Results only include columns that are a match.
- 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.

Note: The header row can contain up to 32,000 characters.

To find the name of a field, you can:

- Use the `describeSObjects()` call in the *SOAP API Developer Guide*, or the `sObject Describe` resource in the *REST API Developer Guide*.
- Use Salesforce Setup.
- Look up the object in *Object Reference*, which lists the field names, types, and descriptions by object.

Use Salesforce Setup to Find Field names

To find an object’s field name in Salesforce Setup:

1. From **Setup**, in the **Quick Find** box, enter *`Object Manager`*. Click **Object Manager**.
2. Click on the object in the list.
3. From the object’s management settings, click on **Fields & Relationships**.
4. Click the field under `Field Label` to find the field name.

For a standard field, use the `Field Name` value as the field column header in your CSV file.

For a custom field, use the `API Name` value as the field column header in a CSV file or the field name identifier in an XML or JSON file. (To find the API Name, click the field name.)

SEE ALSO:

Bulk API 2.0 Older Documentation

Valid Date Format in Records (2.0)

Specify the right format for `dateTime` and `date` fields.

dateTime

Use the *`yyyy-MM-ddTHH:mm:ss.SSS+/-HH:mm`* 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)
- `+/-HH:mm` 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.

date

Use the *`yyyy-MM-dd`* format to specify `date` fields.

Note: Specifying an offset for `date` is not supported.

SEE ALSO:

Bulk API 2.0 Older Documentation

Relationship Fields in a Header Row (2.0)

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 API to get the `relationshipName` property value for a field.

Some objects also have relationships to themselves. For example, the `ReportsTo` 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 `ReportsTo` field for a contact. The `ReportsTo` portion of the column header is the `relationshipName` property value for the `ReportsTo` 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. Because 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.

SEE ALSO:

Bulk API 2.0 Older Documentation

Use Compression for Bulk API 2.0 Responses

For ingest jobs, Bulk API 2.0 can compress the *response body*, which reduces network traffic and improves response time.

Responses are compressed if the client makes a request using the `Accept-Encoding` header, with a value of `gzip`. Bulk API 2.0 compresses the response in gzip format and sends the response to the client with a `Content-Encoding: gzip` *response* header. If a request is made using the `Accept-Encoding` header with a value other than `gzip`, the encoding type is ignored, and the response isn't compressed.

As an example, if a Get Job Successful Record Results on page 49 request is made with the `Accept-Encoding: gzip` header, the response looks something like:

```
HTTP/1.1 200 OK
Date: Tue, 09 Oct 2012 18:36:45 GMT
Content-Type: text/csv; charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked

...compressed response body...
```

Bulk API 2.0 follows the HTTP 1.1 standards for response compression, as described in Using Compression. Most clients automatically support compressed responses. Even though you request a compressed response, the REST framework sometimes doesn't send back the response in a compressed format. Visit `https://developer.salesforce.com/page/Tools` for more information on particular clients.

Troubleshooting Ingest Timeouts

Solve issues encountered with Bulk API 2.0 ingest operations.

To troubleshoot an ingest timeout error, try the following suggestions:

1. **Check your payload.**
   - Check CSV formatting. Prepare CSV Files on page 29
   - Check Date formatting. Valid Date Format in Records (2.0) on page 30
   - Check Relationship fields. Relationship Fields in a Header Row (2.0) on page 31
   - Check that compression is gzip format. Use Compression for Bulk API 2.0 Responses on page 32
   - Check that your data is organized to prevent lock contenttion. Organize Data to Minimize Lock Contention
2. **Create a new ingest job with only the failed and unprocessed records.**
   - To get the failed records, use Get Job Failed Record Results on page 50.
   - To get the unprocessed records, use Get Job Unprocessed Record Results on page 51.
3. **Divide the job into smaller jobs.**
   - Creating smaller requests may help to isolate problems in one or more jobs.
4. **Review your custom logic, such as triggers or flows.**
   - Non-optimized custom logic can contribute to timeouts. In order to speed up insert, update, or delete operations, make sure triggers or flows are optimized. Consider temporarily disabling triggers or flows that are non-essential.

Organize Data to Minimize Lock Contention
To minimize the potential for lock contention, consider pre-organizing your data when planning your Bulk API 2.0 data loads.

Organize Data to Minimize Lock Contention

To minimize the potential for lock contention, consider pre-organizing your data when planning your Bulk API 2.0 data loads.

The Salesforce Platform uses locks to ensure referential integrity of its data — similar to any application built on a relational database. Most transactional database operations only hold these locks for a short time, and the volume isn't significant enough to cause contention. However, when dealing with large data volume objects or processing large jobs, record locks and contention can become an issue.

Tip: Always test your data loads in a sandbox organization first. Processing times can be different in a production organization.

Be Aware of Operations that Increase Lock Contention with Bulk API 2.0

These operations are likely to cause lock contention:

- Creating users
- Updating ownership for records with private sharing
- Updating user roles
- Updating territory hierarchies

Organize Bulk API 2.0 Data to Minimize Lock Contention

For large data loads, sort main records based on their parent record to avoid having different child records (with the same parent) in different batches.

- **Example**: When an `AccountTeamMember` record is created or updated, the corresponding Account for this record is locked during the transaction. If you upload different jobs that include `AccountTeamMember` records, and they all contain references to the same account, *they all try to lock the same account*, and it's likely that you experience a lock timeout. When working with `AccountTeamMember` records, organize your CSV data files by `AccountId`.

Note: Because your data model is unique to your organization, Salesforce can't predict exactly when you experience lock contention problems.

Bulk API 2.0 Automatic Record Lock Handling (PILOT)

Based on a popular suggestion in the Salesforce Idea Exchange, we're now offering a pilot feature that automatically checks for Bulk API 2.0 locking errors and handles them to increase the likelihood of successful job completion.

If you're interested in a pilot to help you automatically minimize lock contention, *contact your account team and ask to be nominated* for the "Bulk API 2.0 Automatic Record Lock Handling Pilot".

Note: This feature is not generally available and is being piloted with certain Customers subject to additional terms and conditions. It is not part of your purchased Services. This feature is subject to change, may be discontinued with no notice at any time in SFDC's sole discretion, and SFDC may never make this feature generally available. Make your purchase decisions only on the basis of generally available products and features. This feature is made available on an AS IS basis and use of this feature is at your sole risk.

Sample CSV Files

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

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"
}
```

You can also associate records using external IDs. For more information, see Upserting Records and Associating with an External ID.

SEE ALSO:

Bulk API 2.0 Older Documentation

Bulk API 2.0 Reference

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

Create a Job

Creates a job representing a bulk operation and its 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.

Upload Job Data

Uploads data for a job using CSV data you provide.

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 more job data. If you abort a job, the job doesn't get queued or processed.

Delete a Job

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

Get All Jobs

Retrieves all jobs in the org.

Get Job Info

Retrieves detailed information about a job.

Get Job Successful Record Results

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

Get Job Failed Record Results

Retrieves a list of failed records for a completed job.

Get Job Unprocessed Record Results

Retrieves a list of unprocessed records for a completed job.

Headers

These are custom HTTP request and response headers that are used for Bulk API 2.0

Create a Job

Creates a job representing a bulk operation and its 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`

Availability

This resource is available in API version 41.0 and later.

Formats

JSON

HTTP Method

POST

Authentication

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

Parameters

None.

Headers

Optionally, use the `Sforce-Call-Options` header to specify a default namespace.

Request Body

Property	Type	Description	Required or Optional
`assignmentRuleId`	string	The ID of an assignment rule to run for a Case or a Lead. The assignment rule can be active or inactive. The ID can be retrieved by using the Lightning Platform SOAP API or the Lightning Platform REST API to query the AssignmentRule object. This property is available in API version 49.0 and later.	Optional
`columnDelimiter`	ColumnDelimiterEnum	The column delimiter used for CSV job data. The default value is `COMMA`. Valid values are: • `BACKQUOTE`—backquote character (`)	Optional

Property	Type	Description	Required or Optional
		• `CARET`—caret character (^) • `COMMA`—comma character (,) which is the default delimiter • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character	
`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
`operation`	OperationEnum	The processing operation for the job. Valid values are: • `insert` • `delete` • `hardDelete` • `update` • `upsert` Note: When the `hardDelete` value is specified, the deleted records aren't stored in the Recycle Bin. Instead, they become immediately eligible for deletion. The permission for this operation, “Bulk API Hard Delete,” is disabled by default and must be enabled by an administrator. A Salesforce user license is required for hard delete.	Required

For multipart requests, the request body can also include CSV record data. See Usage Notes on page 40 for more details.

Response Body

Property	Type	Description
`apiVersion`	string	The API version that the job was created in.

Property	Type	Description
`assignmentRuleId`	id	The ID of the assignment rule. This property is only shown if an assignment rule is specified when the job is created.
`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	For future use. How the request was processed. Currently only parallel mode is supported. (When other modes are added, the mode will be chosen automatically by the API and will not be user configurable.)
`contentType`	ContentType	The format of the data being processed. Only CSV is supported.
`contentUrl`	URL	The URL to use for Upload Job Data on page 40 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 • `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` • `hardDelete` • `update` • `upsert`

Property	Type	Description
`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`—Some records in the job 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 (100,000 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--
```

SEE ALSO:

Bulk API 2.0 Older Documentation

Upload Job Data

Uploads data for a job using CSV data you provide.

URI

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

Availability

This resource is available in API version 41.0 and later.

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

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.

SEE ALSO:

Bulk API 2.0 Older Documentation

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 more job data. If you abort a job, the job doesn't get queued or processed.

URI

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

Availability

This resource is available in API version 41.0 and later.

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.
`assignmentRuleId`	id	The ID of the assignment rule. This property is only shown if an assignment rule is specified when the job is created.
`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	For future use. How the request was processed. Currently only parallel mode is supported. (When other modes are added, the mode will be chosen automatically by the API and will not be user configurable.)
`contentType`	ContentType	The format of the data being processed. Only CSV is supported.
`contentUrl`	URL	The URL to use for Upload Job Data on page 40 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 • `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

Property	Type	Description
`object`	string	The object type for the data being processed.
`operation`		The processing operation for the job. Values include: • `insert` • `delete` • `hardDelete` • `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`—Some records in the job 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.

SEE ALSO:

Bulk API 2.0 Older Documentation

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`***

Availability

This resource is available in API version 41.0 and later.

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.

SEE ALSO:

Bulk API 2.0 Older Documentation

Get All Jobs

Retrieves all jobs in the org.

URI

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

Availability

This resource is available in API version 41.0 and later.

Formats

JSON

HTTP Method

GET

Authentication

`Authorization: Bearer token`

Parameters

Parameter	Description
`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.

Property	Type	Description
`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	For future use. How the request was processed. Currently only parallel mode is supported. (When other modes are added, the mode will be chosen automatically by the API and will not be user configurable.)
`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 • `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.

Property	Type	Description
`operation`		The processing operation for the job. Values include: • `insert` • `delete` • `hardDelete` • `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`—Some records in the job 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.

SEE ALSO:

Bulk API 2.0 Older Documentation

Get Job Info

Retrieves detailed information about a job.

URI

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

Availability

This resource is available in API version 41.0 and later.

Formats

JSON

HTTP Method

GET

Authentication

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

Request parameters

Parameter	Description	Required or Optional
`jobId`	The ID of the job.	Required

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.
`assignmentRuleId`	string	The ID of an assignment rule to run for a Case or a Lead. .
`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	For future use. How the request was processed. Currently only parallel mode is supported. (When other modes are added, the mode will be chosen automatically by the API and will not be user configurable.)
`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 • `Classic`: Bulk API 1.0 job • `V2Ingest`: Bulk API 2.0 job

Property	Type	Description
`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`	long	The number of records that were not processed successfully in this job. This property is of type int in API version 46.0 and earlier.
`numberRecordsProcessed`	long	The number of records already processed. This property is of type int in API version 46.0 and earlier.
`object`	string	The object type for the data being processed.
`operation`	OperationEnum	The processing operation for the job. Values include: • `insert` • `delete` • `hardDelete` • `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. • `InProgress`: The job is being processed by Salesforce. This includes automatic optimized chunking of job data and processing of job operations. • `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`: Some records in the job 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.

Response Body - For an Unsuccessful Request

If the request fails, the server returns a non-200 status, and the request body shows details of the error. For example, if the job has been deleted the status is 404 (Not Found) and the response body is:

```
[{
 "errorCode": "NOT_FOUND",
 "message": "The requested resource does not exist"
}]
```

Example

This example gets information about the job with ID `7506g00000DhRA2AAN`:

```
curl --include --request GET \
--header "Authorization: Bearer token" \
"https://instance.salesforce.com/services/data/vXX.X/jobs/query/7506g00000DhRA2AAN
```

The response is:

```
{
  "id" : "7506g00000DhRA2AAN",
  "operation" : "insert",
  "object" : "Account",
  "createdById" : "0056g000005HQPyAAO",
  "createdDate" : "2018-12-18T22:51:36.000+0000",
  "systemModstamp" : "2018-12-18T22:51:58.000+0000",
  "state" : "Open",
  "concurrencyMode" : "Parallel",
  "contentType" : "CSV",
  "apiVersion" : 55.0,
  "jobType" : "V2Ingest",
  "contentUrl" : "services/data/v55.0/jobs/ingest/7506g00000DhRA2AAN/batches",
  "lineEnding" : "LF",
  "columnDelimiter" : "COMMA",
  "retries" : 0,
  "totalProcessingTime" : 0,
  "apiActiveProcessingTime" : 0,
  "apexProcessingTime" : 0
}
```

SEE ALSO:

Bulk API 2.0 Older Documentation

Get Job Successful Record Results

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

URI

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

Availability

This resource is available in API version 41.0 and later.

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.
- Results are not recorded for batches that exceed the daily batch allocation.

SEE ALSO:
Bulk API 2.0 Older Documentation

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/`

Availability
This resource is available in API version 41.0 and later.

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.
- Results are not recorded for batches that exceed the daily batch allocation.

SEE ALSO:

Bulk API 2.0 Older Documentation

Get Job Unprocessed Record Results

Retrieves a list of unprocessed records for a completed job.

URI

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

Availability

This resource is available in API version 41.0 and later.

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 an unprocessed record and contains this 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.
- Results are not recorded for batches that exceed the daily batch allocation.

SEE ALSO:

Bulk API 2.0 Older Documentation

Headers

These are custom HTTP request and response headers that are used for Bulk API 2.0

Sforce Call Options Header
Use the Sforce Call Options header to specify client-specific options when accessing Bulk API 2.0 resources.

Warnings Header
This header is returned if there are warnings, such as the use of a deprecated version of the API.

Content Type Header
Use the Content Type header to specify the format for your Bulk API 2.0 request. Set the value of this header to match the `contentType` of the body of your request.

Line Ending Header (Ignored)
Bulk API 2.0 does not honor the `Sforce-Line-Ending` header sent in job creation requests. Bulk API 2.0 ignores this header.

SEE ALSO:

Bulk API 2.0 Older Documentation

Sforce Call Options Header

Use the Sforce Call Options header to specify client-specific options when accessing Bulk API 2.0 resources.

Header Field Name and Values

Field name

`Sforce-Call-Options`

Field values

- `client`—A string that identifies a client, for use, for example, in event log files.

- `defaultNamespace`—A string that identifies a developer namespace prefix. Resolve field names in managed packages without having to specify the namespace everywhere.

Example

If the developer namespace prefix is `battle`, and you have a custom field called `botId` in a package, set the default namespace with the call options header:

```
Sforce-Call-Options: client=CaseSensitiveToken, defaultNamespace=battle
```

Then queries such as the following succeed:

```
/services/data/vXX.X/query/?q=SELECT+Id+botID__c+FROM+Account
```

In this case, the actual field queried is the `battle__botId__c` field.

Using this header allows you to write client code without having to specify the namespace prefix. In the previous example, without the header you must write `battle__botId__c`.

If this field is set, and the query also specifies the namespace, the response doesn't include the prefix. For example, if you set this header to `battle`, and issue a query like `SELECT+Id+battle__botID__c+FROM+Account`, the response uses a `botId__c` element, not a `battle_botId__c` element.

The `defaultNamespace` field is ignored when retrieving describe information, which avoids ambiguity between namespace prefixes and customer fields of the same name.

SEE ALSO:

Bulk API 2.0 Older Documentation

Warnings Header

This header is returned if there are warnings, such as the use of a deprecated version of the API.

Header Field Name and Values

Field name

`Warning`

Field values

- `warningCode`
- `warningMessage`

For warnings about deprecated API versions, the `warningCode` is 299.

Example

```
Warning: 299 - "This API is deprecated and will be removed by Summer '22. Please see
https://help.salesforce.com/articleView?id=000351312 for details."
```

SEE ALSO:

Bulk API 2.0 Older Documentation

Content Type Header

Use the Content Type header to specify the format for your Bulk API 2.0 request. Set the value of this header to match the `contentType` of the body of your request.

Header Field Name and Values

Field name

`Content-Type`

Field values

- `text/csv`
- application/json
- application/xml

Example

`Content-Type: application/json`

Line Ending Header (Ignored)

Bulk API 2.0 does not honor the `Sforce-Line-Ending` header sent in job creation requests. Bulk API 2.0 ignores this header.

Bulk API 2.0 Query

Bulk query jobs enable asynchronous processing of SOQL queries. They're designed to handle queries that return large amounts of data (10,000 records or more).

Understanding Bulk API 2.0 Query
Learn about Bulk API 2.0 availability, supported methods, and limits.

Create a Query Job
Creates a query job.

Get Information About a Query Job
Gets information about one query job.

Get Results for a Query Job
Gets the results for a query job. The job must have the state `JobComplete`.

Abort a Query Job
Aborts a query job.

Delete a Query Job
Deletes a query job. When a job is deleted, job data stored by Salesforce is deleted and job metadata information is removed. The job no longer displays in the Bulk Data Load Jobs page in Salesforce.

Get Information About All Query Jobs
Gets information about all query jobs in the org. The information includes Bulk API 2.0 query jobs and all Bulk API jobs.

Troubleshooting Query Timeouts
To troubleshoot timeout errors in Bulk API 2.0 query operations, apply additional filter criteria.

Understanding Bulk API 2.0 Query

Learn about Bulk API 2.0 availability, supported methods, and limits.

Availability

Query jobs in Bulk API 2.0 are available in API version 47.0 and later.

Supported URIs and Methods

This table lists the URIs and methods supported by queries in Bulk API 2.0.

URI	HTTP Method	Description
`/services/data/vXX.X/jobs/query`	POST	Creates a query job.
`/services/data/vXX.X/jobs/query`	GET	Gets information about all query jobs in the org.
`/services/data/vXX.X/jobs/query/queryJobId`	GET	Gets information about one query job.
`/services/data/vXX.X/jobs/query/queryJobId/results`	GET	Gets the results for a query job.
`/services/data/vXX.X/jobs/query/queryJobId`	PATCH	Aborts a query job.
`/services/data/vXX.X/jobs/query/queryJobId`	DELETE	Deletes a query job.

Limits

Bulk API 2.0 doesn’t support SOQL queries that include any of these items:

- GROUP BY, OFFSET, or TYPEOF clauses.
- Aggregate Functions such as `COUNT()`.
- Date functions in GROUP BY clauses. (Date functions in WHERE clauses are supported.)
- Compound address fields or compound geolocation fields. (Instead, query the individual components of compound fields.)
- Parent-to-child relationship queries. (Child-to-parent relationship queries are supported.)

This table lists the limits of query jobs in Bulk API 2.0.

Item	Limit
The maximum number of query jobs per 24-hour rolling window). The current number can be seen in the `DailyBulkV2QueryJobs` value in the response to the `/vXX.X/limits/` REST API method.	10,000

Item	Limit
The maximum query result size that can be stored in a 24 hour rolling window. The current size can be seen in the `DailyBulkV2QueryFileStorageMB` value in the response to the `/vXX.X/limits/` REST API method.	1 TB

SEE ALSO:

Bulk API 2.0 Older Documentation

Create a Query Job

Creates a query job.

Syntax

URI

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

Available since release

This resource is available in API version 47.0 and later.

Format

application/json

HTTP method

POST

Authentication

Authorization:Bearer *token*

Headers

Optionally, use the `Sforce-Call-Options` header to specify a default namespace.

Request body

The request body specifies the query to be performed.

```
{
  "operation": "query",
  "query": "SELECT Id FROM Account"
}
```

Note: Bulk API 2.0 doesn't support SOQL queries that include any of these items:

- GROUP BY, OFFSET, or TYPEOF clauses.
- Aggregate Functions such as `COUNT()`.
- Date functions in GROUP BY clauses. (Date functions in WHERE clauses are supported.)
- Compound address fields or compound geolocation fields. (Instead, query the individual components of compound fields.)
- Parent-to-child relationship queries. (Child-to-parent relationship queries are supported.)

You can also specify some optional parameters. For example:

```
{
  "operation" : "query",
  "query" : "SELECT Id FROM Account",
  "contentType" : "CSV",
  "columnDelimiter" : "CARET",
  "lineEnding" : "CRLF"
}
```

Request parameters

Parameter	Description	Required or Optional
`operation`	The type of query. Possible values are: • `query`—Returns data that hasn't been deleted or archived. For more information, see query() in *SOAP API Developer Guide*. • `queryAll`—Returns records that have been deleted because of a merge or delete, and returns information about archived Task and Event records. For more information, see queryAll() in *SOAP API Developer Guide*.	Required
`query`	The query to be performed.	Required
`contentType`	The format to be used for the results. Currently the only supported value is `CSV` (comma-separated variables). Defaults to `CSV`. Note: The actual separator can be a character other than a comma. The `columnDelimiter` parameter specifies what character to use.	Optional
`columnDelimiter`	The column delimiter used for CSV job data. The default value is `COMMA`. Possible values are: • `BACKQUOTE`—back quote character (\`) • `CARET`—caret character (^) • `COMMA`—comma character (,) • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character	Optional
`lineEnding`	The line ending used for CSV job data, marking the end of a data row. The default is `LF`. Possible values are: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character	Optional

Response Body

```
{
   "id" : "750R0000000zlh9IAA",
   "operation" : "query",
   "object" : "Account",
   "createdById" : "005R0000000GiwjIAC",
   "createdDate" : "2018-12-10T17:50:19.000+0000",
   "systemModstamp" : "2018-12-10T17:50:19.000+0000",
   "state" : "UploadComplete",
   "concurrencyMode" : "Parallel",
   "contentType" : "CSV",
   "apiVersion" : 46.0,
   "lineEnding" : "LF",
   "columnDelimiter" : "COMMA"
}
```

Response Parameters

Parameter	Description
`id`	The unique ID for this job.
`operation`	The type of query.
`object`	The object type being queried.
`createdById`	The ID of the user who created the job.
`createdDate`	The UTC date and time when the job was created.
`systemModstamp`	The UTC date and time when the API last updated the job information.
`state`	The current state of processing for the job. Possible values are: • `UploadComplete`—All job data has been uploaded and the job is ready to be processed. Salesforce has put the job in the queue. • `InProgress`—Salesforce is processing the job. • `Aborted`—The job has been aborted. See Abort a Query Job. • `JobComplete`—Salesforce has finished processing the job. • `Failed`—The job failed.
`concurrencyMode`	Reserved for future use. How the request is processed. Currently only parallel mode is supported. (When other modes are added, the API chooses the mode automatically. The mode isn't user configurable.)
`contentType`	The format to be used for the results. Currently the only supported value is `CSV`.

Parameter	Description
`apiVersion`	The API version that the job was created in.
`lineEnding`	The line ending used for CSV job data, marking the end of a data row.
`columnDelimiter`	The column delimiter used for CSV job data.

Example

This example creates a job that queries Accounts.

```
curl --include --request POST \
--header "Authorization: Bearer token" \
--header "Accept: application/json " \
--header "Content-Type: application/json" \
--data '{
  "operation": "query",
  "query": "SELECT Id, Name FROM Account"
}' \
https://instance.salesforce.com/services/data/vXX.X/jobs/query
```

The response is:

```
HTTP/1.1 200 OK
{
   "id" : "750R0000000zw4yIAA",
   "operation" : "query",
   "object" : "Account",
   "createdById" : "005R0000000GiwjIAC",
   "createdDate" : "2018-12-17T21:00:17.000+0000",
   "systemModstamp" : "2018-12-17T21:00:17.000+0000",
   "state" : "UploadComplete",
   "concurrencyMode" : "Parallel",
   "contentType" : "CSV",
   "apiVersion" : 46.0,
   "lineEnding" : "LF",
   "columnDelimiter" : "COMMA"
}
```

SEE ALSO:

Bulk API 2.0 Older Documentation

Get Information About a Query Job

Gets information about one query job.

Syntax

URI

`/services/data/vXX.X/jobs/query/queryJobId`

Available since release

This resource is available in API version 47.0 and later.

Formats

JSON

HTTP methods

GET

Authentication

Authorization: Bearer *`token`*

Request parameters

Parameter	Description	Required or Optional
`queryJobId`	The ID of the query job.	Required

Response Body

```
{
   "id" : "750R0000000zlh9IAA",
   "operation" : "query",
   "object" : "Account",
   "createdById" : "005R0000000GiwjIAC",
   "createdDate" : "2018-12-10T17:50:19.000+0000",
   "systemModstamp" : "2018-12-10T17:51:27.000+0000",
   "state" : "JobComplete",
   "concurrencyMode" : "Parallel",
   "contentType" : "CSV",
   "apiVersion" : 46.0,
   "jobType" : "V2Query",
   "lineEnding" : "LF",
   "columnDelimiter" : "COMMA",
   "numberRecordsProcessed" : 500,
   "retries" : 0,
   "totalProcessingTime" : 334
}
```

Response Parameters

Parameter	Type	Description
`id`	string	The unique ID for this job.
`operation`	OperationEnum	The type of query. Possible values are: • `query`—Returns data that hasn't been deleted or archived. For more information, see query() in *SOAP API Developer Guide*. • `queryAll`—Returns records that have been deleted because of a merge

Parameter	Type	Description
		or delete, and returns information about archived Task and Event records. For more information, see queryAll() in *SOAP API Developer Guide*.
`object`	string	The object type being queried.
`createdById`	string	The ID of the user who created the job.
`createdDate`	dateTime	The UTC date and time when the job was created.
`systemModstamp`	dateTime	The UTC date and time when the API last updated the job information.
`state`	JobStateEnum	The current state of processing for the job. Possible values are: • `UploadComplete`—All job data has been uploaded and the job is ready to be processed. Salesforce has put the job in the queue. • `InProgress`—Salesforce is processing the job. • `Aborted`—The job has been aborted. See Abort a Query Job. • `JobComplete`—Salesforce has finished processing the job. • `Failed`—The job failed.
`concurrencyMode`	ConcurrencyModeEnum	Reserved for future use. How the request is processed. Currently only parallel mode is supported. (When other modes are added, the API chooses the mode automatically. The mode isn't user configurable.)
`contentType`	ContentType	The format that is used for the results. Currently the only supported value is `CSV`.
`apiVersion`	string	The API version that the job was created in.
`jobType`	JobTypeEnum	The job's type. For a query job, the type is always `V2Query`.

Parameter	Type	Description
`lineEnding`	LineEndingEnum	The line ending used for CSV job data, marking the end of a data row. The default is `LF`. Possible values are: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character
`columnDelimiter`	ColumnDelimiterEnum	The column delimiter used for CSV job data. The default value is `COMMA`. Possible values are: • `BACKQUOTE`—back quote character (`) • `CARET`—caret character (^) • `COMMA`—comma character (,) • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character
`numberRecordsProcessed`	long	The number of records processed in this job.
`retries`	int	The number of times that Salesforce attempted to save the results of an operation. Repeated attempts indicate a problem such as a lock contention.
`totalProcessingTime`	long	The number of milliseconds taken to process the job.

Response Body - For an Unsuccessful Request

If the request fails, the server returns a non-200 status, and the request body shows details of the error. For example, if the job has been deleted the status is 404 (Not Found) and the response body is:

```
[{
 "errorCode": "NOT_FOUND",
 "message": "The requested resource does not exist"
}]
```

Example

This example gets information about the job with ID `750R0000000zxikIAA`:

```
curl --include --request GET \
--header "Authorization: Bearer token" \
"https://instance.salesforce.com/services/data/vXX.X/jobs/query/750R0000000zxikIAA
```

The response is:

```
{
   "id" : "750R0000000zxikIAA",
   "operation" : "query",
   "object" : "Account",
   "createdById" : "005R0000000GiwjIAC",
   "createdDate" : "2018-12-18T22:51:36.000+0000",
   "systemModstamp" : "2018-12-18T22:51:58.000+0000",
   "state" : "JobComplete",
   "concurrencyMode" : "Parallel",
   "contentType" : "CSV",
   "apiVersion" : 46.0,
   "jobType" : "V2Query",
   "lineEnding" : "LF",
   "columnDelimiter" : "COMMA",
   "numberRecordsProcessed" : 740003,
   "retries" : 0,
   "totalProcessingTime" : 21046
}
```

SEE ALSO:

Bulk API 2.0 Older Documentation

Get Results for a Query Job

Gets the results for a query job. The job must have the state `JobComplete`.

Syntax

URI

`/services/data/vXX.X/jobs/query/queryJobId/results`

or

```
/services/data/vXX.X/jobs/query/queryJobId/results
   ?locator=locator
   &maxRecords=maxRecords
```

Note: Use the same API version to get query results that you used to create the query. Otherwise, the call returns a 409 error.

Available since release

This resource is available in API version 47.0 and later.

Formats

CSV

HTTP methods

GET

Authentication

Authorization: Bearer *token*

Request parameters

Parameter	Description	Required or Optional
`queryJobId`	The ID of the query job.	Required
`locator`	A string that identifies a specific set of query results. Providing a value for this parameter returns only that set of results. Omitting this parameter returns the first set of results. You can find the locator string for the next set of results in the response of each request. See Example and Rules and Guidelines. As long as the associated job exists, the locator string for a set of results does not change. You can use the locator to retrieve a set of results multiple times.	Optional
`maxRecords`	The maximum number of records to retrieve per set of results for the query. The request is still subject to the size limits. If you are working with a very large number of query results, you may experience a timeout before receiving all the data from Salesforce. To prevent a timeout, specify the maximum number of records your client is expecting to receive in the `maxRecords` parameter. This splits the results into smaller sets with this value as the maximum size. If you don't provide a value for this parameter, the server uses a default value based on the service.	Optional

Response Body

If the request is successful, the status code is 200 (OK) and the request body contains the results of the job's query. For example:

```
"Id","Name"
"005R0000000UyrWIAS","Jane Dunn"
"005R0000000GiwjIAC","George Wright"
"005R0000000GiwoIAC","Pat Wilson"
...
```

Note: In API version 50.0 and later, the order of the columns returned by the query is the same as the order you requested them. In version 49.0 and earlier, the order of the columns is returned alphabetically.

Response Headers

Header	Description
`Sforce-NumberOfRecords`	The number of records in this set.

Header	Description
`Sforce-Locator`	The locator for the next set of results (if there are any). Use this value in other `GET` request to retrieve the next set of query results. This value is a pseudo random string (for example, `MTAwMDA`). The length of this string varies depending on how many sets of results there are. If there are no more sets of query results, this value is the string 'null'. See Example and Rules and Guidelines.

Example

This example retrieves the results for the job with ID `750R0000000zxr8IAA`. It also shows how to use the `locator` and `maxRecords` query parameters. (In this example, we use them both, but they are independent. You don't have to use them together.)

We start by sending an initial request to retrieve the first set of query results. We don't use the `locator` parameter because we want to get the first set of results.

```
curl --include --request GET \
--header "Authorization: Bearer token" \
--header "Accept: text/csv" \
https://instance.salesforce.com/services/data/vXX.X/jobs/query/750R0000000zxr8IAA/results
?maxRecords=50000
```

Note: The `Accept` header must match what was specified when the job was created. Currently, only `text/csv` is supported.

The response body is:

```
HTTP/1.1 200 OK
...
Sforce-Locator: MTAwMDA
Sforce-NumberOfRecords: 50000
...

"Id","Name"
"005R0000000UyrWIAS","Jane Dunn"
"005R0000000GiwjIAC","George Wright"
"005R0000000GiwoIAC","Pat Wilson"
...
```

The response includes `MTAwMDA` as a value for the `Sforce-Locator` header. This value is not 'null', which means that there are more query results that we can retrieve.

To retrieve the next set of query results, we send another request, using the `locator` parameter and the locator string, `MTAwMDA`.

```
curl --include --request GET \
--header "Accept: text/csv" \
https://instance.salesforce.com/services/data/vXX.X/jobs/query/750R0000000zxr8IAA/results
?locator=MTAwMDA&maxRecords=50000
```

The response body is:

```
HTTP/1.1 200 OK
...
Sforce-Locator: MjAwMDAw
Sforce-NumberOfRecords: 50000
...

"Id","Name"
"005R0000000UyrWIAv","James Wu"
"005R0000000GiwjIxx","Samantha Jones"
"005R0000000GiwoIAB","Doug West"
...
```

Notice that the locator value has changed. This means that there is another set of query results that we can retrieve. We use the new locator string, `MjAwMDAw`, in another request.

```
curl --include --request GET \
--header "Accept: text/csv" \
https://instance.salesforce.com/services/data/vXX.X/jobs/query/750R0000000zxr8IAA/results
?locator=MjAwMDAw&maxRecords=50000
```

We repeat this process until the value of the `Sforce-Locator` header is 'null', which indicates that there are no more results to retrieve.

```
HTTP/1.1 200 OK

Sforce-Locator: null
Sforce-NumberOfRecords: 6155
...

"Id","Name"
...
```

Rules and Guidelines

To retrieve your full set of query results, follow these rules and guidelines.

1. Use `/services/data/vXX.X/jobs/query/queryJobId/results` to get the first set of results for the job.
2. If there are no more results, the `Sforce-Locator` header's value is the string 'null'. Otherwise, set the `locator` query parameter to that value to get the next set of results.

   Note: For `locator`, use only the value from the `Sforce-Locator` header. Don't try to guess what it is. How this parameter is evaluated is subject to change.

3. Repeat this process until the `Sforce-Locator` header's value is the string 'null'. That set is the last set of results.

SEE ALSO:

Bulk API 2.0 Older Documentation

Abort a Query Job

Aborts a query job.

Note:

- To abort a job, you must be the job's creator or have the Manage Data Integrations permission.
- You can only abort jobs that are in the following states:
  - `UploadComplete`
  - `InProgress`

Syntax

URI

`/services/data/v`***`XX.X/`***`jobs/query/`***`queryJobId`***

Available since release

This resource is available in API version 47.0 and later.

Formats

JSON

HTTP methods

PATCH

Authentication

Authorization: Bearer *`token`*

Request body

The request body must be the following:

```
{
  "state": "Aborted"
}
```

Request parameters

Parameter	Description	Required or Optional
`queryJobId`	The ID of the query job to be deleted.	Required

Response Body

If the request is successful, the response is similar to that for Get Results for a Query Job but the `state` is `Aborted`. For example:

```
{
   "id" : "750R000000146UvIAI",
   "operation" : "query",
   "object" : "Account",
   "createdById" : "005R0000000GiwjIAC",
   "createdDate" : "2018-12-18T16:15:31.000+0000",
   "systemModstamp" : "2018-12-18T16:15:32.000+0000",
   "state" : "Aborted",
   "concurrencyMode" : "Parallel",
   "contentType" : "CSV",
   "apiVersion" : 46.0
}
```

Response Parameters

Parameter	Description
`id`	The unique ID for this job.
`operation`	The type of query. Possible values are: • `query`—Returns data that hasn't been deleted or archived. For more information, see query() in *SOAP API Developer Guide*. • `queryAll`—Returns records that have been deleted because of a merge or delete, and returns information about archived Task and Event records. For more information, see queryAll() in *SOAP API Developer Guide*.
`object`	The object type being queried.
`createdById`	The ID of the user who created the job.
`createdDate`	The UTC date and time when the job was created.
`systemModstamp`	The UTC date and time when the API last updated the job information.
`state`	The current state of processing for the job. Possible values are: • `UploadComplete`—All job data has been uploaded and the job is ready to be processed. Salesforce has put the job in the queue. • `InProgress`—Salesforce is processing the job. • `Aborted`—The job has been aborted. See Abort a Query Job. • `JobComplete`—Salesforce has finished processing the job. • `Failed`—The job failed.
`concurrencyMode`	Reserved for future use. How the request is processed. Currently only parallel mode is supported. (When other modes are added, the API chooses the mode automatically. The mode isn't user configurable.)
`contentType`	The format that is used for the results. Currently the only supported value is `CSV`.
`apiVersion`	The API version that the job was created in.

Response Body - For an Unsuccessful Request

If the request fails, the server returns a non-200 status, and the request body shows details of the error. For example:

```
HTTP/1.1 400 Bad Request
[{
 "errorCode": "INVALIDJOBSTATE",
 "message": "Aborting already Completed Job not allowed"
}]
```

Example

This example aborts the job with ID `750R000000146UvIAI`:

```
curl  --request PATCH \
--header "Authorization: Bearer token" \
--header "Content-Type: application/json" \
--data '{
  "state": "Aborted"
}' \
https://instance.salesforce.com/services/data/vXX.X/jobs/query/750R000000146UvIAI
```

The response is:

```
{
  "id": "750R000000146UvIAI",
  "operation": "query",
  "object": "Account",
  "createdById": "005R0000000GiwjIAC",
  "createdDate": "2018-12-18T20:51:39.000+0000",
  "systemModstamp": "2018-12-18T20:51:41.000+0000",
  "state": "Aborted",
  "concurrencyMode": "Parallel",
  "contentType": "CSV",
  "apiVersion": 46.0
}
```

SEE ALSO:

Bulk API 2.0 Older Documentation

Delete a Query Job

Deletes a query job. When a job is deleted, job data stored by Salesforce is deleted and job metadata information is removed. The job no longer displays in the Bulk Data Load Jobs page in Salesforce.

Note: You can only delete a job if its state is `JobComplete`, `Aborted`, or `Failed`.

Syntax

URI

`/services/data/vXX.X/jobs/query/queryJobId`

Available since release

This resource is available in API version 47.0 and later.

Formats

None

HTTP methods

DELETE

Authentication

Authorization: Bearer *token*

Request parameters

Parameter	Description	Required or Optional
`queryJobId`	The ID of the query job to be deleted.	Required

Response Body

If the method is successful, the status code is 204 (No Content) and there is no response body.

Response Body - For an Unsuccessful Request

If the request fails, the server returns a 400 (Bad Request) status, and the request body shows details of the error. For example:

```
HTTP/1.1 400 Bad Request
[{
 "errorCode": "API_ERROR",
 "message": "Error encountered when deleting the job because the job is not terminated"
}]
```

Example

This example deletes the job with ID `750R0000000zxnaIAA`.

```
curl --include --request DELETE \
--header "Authorization: Bearer token \
--header "Content-Type: " \
https://instance.salesforce.com/services/data/vXX.X/jobs/query/750R0000000zxnaIAA
```

The response status is

```
204 No Content
```

SEE ALSO:

Bulk API 2.0 Older Documentation

Get Information About All Query Jobs

Gets information about all query jobs in the org. The information includes Bulk API 2.0 query jobs and all Bulk API jobs.

Syntax

URI

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

```
/services/data/vXX.X/jobs/query/
   ?isPkChunkingEnabled=isPkChunkingEnabled
   &jobType=jobType
   &concurrencyMode=concurrencyMode
   &queryLocator=queryLocator
```

Available since release

This resource is available in API version 47.0 and later.

Formats

JSON

HTTP methods

GET

Authentication

Authorization: Bearer *token*

Request parameters

Parameter	Description	Required or Optional
`isPkChunkingEnabled`	If set to true, the request only returns information about jobs where PK Chunking is enabled. This only applies to Bulk API (not Bulk API 2.0) jobs. For more information on PK Chunking, see Use PK Chunking to Extract Large Data Sets from Salesforce.	Optional
`jobType`	Gets information only about jobs matching the specified job type. Possible values are: • `Classic`—Bulk API jobs. This includes both query jobs and ingest jobs. • `V2Query`—Bulk API 2.0 query jobs. • `V2Ingest`—Bulk API 2.0 ingest (upload and upsert) jobs.	Optional
`concurrencyMode`	For future use. Gets information only about jobs matching the specified concurrency mode. Possible values are `serial` and `parallel`. Note: Currently only parallel mode is supported.	Optional
`queryLocator`	Gets information about jobs starting with that locator value. Note: Do not enter your own value here. Always use the value from the `nextRecordsUrl` from the previous set. See Example and Rules and Guidelines.	Optional

Response Body

The response contains a completion flag, an array of records, and a locator value to be used to obtain more records. For example:

```
{
   "done" : false,
   "records" : [
      {
         "id" : "750R0000000zhfdIAA",
         "operation" : "query",
         "object" : "Account",
         "createdById" : "005R0000000GiwjIAC",
         "createdDate" : "2018-12-07T19:58:09.000+0000",
         "systemModstamp" : "2018-12-07T19:59:14.000+0000",
         "state" : "JobComplete",
         "concurrencyMode" : "Parallel",
         "contentType" : "CSV",
         "apiVersion" : 55.0,
         "jobType" : "V2Query",
         "lineEnding" : "LF",
         "columnDelimiter" : "COMMA"
      },
      {
         "id" : "750R0000000zhjzIAA",
         "operation" : "query",
         "object" : "Account",
         "createdById" : "005R0000000GiwjIAC",
         "createdDate" : "2018-12-07T20:52:28.000+0000",
         "systemModstamp" : "2018-12-07T20:53:15.000+0000",
         "state" : "JobComplete",
         "concurrencyMode" : "Parallel",
         "contentType" : "CSV",
         "apiVersion" : 55.0,
         "jobType" : "V2Query",
         "lineEnding" : "LF",
         "columnDelimiter" : "COMMA"
      },
     ...
   ],
   "nextRecordsUrl" :
"/services/data/v55.0/jobs/ingest?queryLocator=01gR0000000opRTIAY-2000"
}
```

Response Parameters

Parameter	Description
`done`	This is `true` if this is the last (or only) set of results. It is `false` if there are more records to fetch. See Example and Rules and Guidelines.
`records`	An array of record objects.
`nextRecordsUrl`	The URI to get the next set of records (if there are any).

Parameter	Description
	This method returns up to 1,000 result rows per request. If there are more than 1,000 records, use the `nextRecordsUrl` to get the next set of records. See Example and Rules and Guidelines. This parameter is `null` if there are no more records to fetch.

Record Objects

Parameter	Description
`id`	The unique ID for this job.
`operation`	The type of query. Possible values are: • `query`—Returns data that hasn't been deleted or archived. For more information, see query() in *SOAP API Developer Guide*. • `queryAll`—Returns records that have been deleted because of a merge or delete, and returns information about archived Task and Event records. For more information, see queryAll() in *SOAP API Developer Guide*.
`object`	The object type being queried.
`createdById`	The ID of the user who created the job.
`createdDate`	The UTC date and time when the job was created.
`systemModstamp`	The UTC date and time when the API last updated the job information.
`state`	The current state of processing for the job. Possible values are: • `UploadComplete`—All job data has been uploaded and the job is ready to be processed. Salesforce has put the job in the queue. • `InProgress`—Salesforce is processing the job. • `Aborted`—The job has been aborted. See Abort a Query Job. • `JobComplete`—Salesforce has finished processing the job. • `Failed`—The job failed.
`concurrencyMode`	Reserved for future use. How the request is processed. Currently only parallel mode is supported. (When other modes are added, the API chooses the mode automatically. The mode isn't user configurable.)

Parameter	Description
`contentType`	The format to be used for the results. Currently the only supported value is `CSV` (comma-separated variables). Defaults to `CSV`. Note: The actual separator can be a character other than a comma. The `columnDelimiter` parameter specifies what character to use.
`apiVersion`	The API version that the job was created in.
`lineEnding`	The line ending used for CSV job data, marking the end of a data row. The default is `LF`. Possible values are: • `LF`—linefeed character • `CRLF`—carriage return character followed by a linefeed character
`columnDelimiter`	The column delimiter used for CSV job data. The default value is `COMMA`. Possible values are: • `BACKQUOTE`—back quote character (`) • `CARET`—caret character (^) • `COMMA`—comma character (,) • `PIPE`—pipe character (|) • `SEMICOLON`—semicolon character (;) • `TAB`—tab character

Example

This example shows how to use the `nextRecordsUrl` query parameter.

The first request doesn't use `nextRecordsUrl`, because we don't know what value to use yet.

```
curl --request GET \
--header "Authorization: Bearer token"
https://instance.salesforce.com/services/data/vXX.X/jobs/query
```

The response body is:

```
{
    "done": false,
    "nextRecordsUrl":
"/services/data/vXX.X/jobs/ingest?queryLocator=01gRM000000NS1vYAG-1000",
    "records": [
        {
            ...
        }
    ]
}
```

Because there are more records than can be returned in a single response, the value of `done` in the response isn't `true`. We use the value of `nextRecordsUrl`, `/services/data/vXX.X/jobs/ingest?queryLocator=01gRM000000NS1vYAG-1000`, as the URI to get the next set of records:

```
curl --request GET \
--header "Authorization: Bearer token"
https://instance.salesforce.com/services/data/vXX.X/jobs/query?queryLocator=01gRM000000NS1vYAG-1000
```

Repeat this process until `done` is `true`, indicating that there are no more results to fetch.

Rules and Guidelines

To use this URI, follow these rules and guidelines.

1. Use `/services/data/vXX.X/jobs/query` to get the first set of records.
2. If there are more records than can be listed, the response body has `done` set to `false`. Use the value of `nextRecordsUrl` to get the next set of records.
3. Repeat this process until `done` is `true`. That set is the last set of records.

SEE ALSO:

Bulk API 2.0 Older Documentation

Troubleshooting Query Timeouts

To troubleshoot timeout errors in Bulk API 2.0 query operations, apply additional filter criteria.

SEE ALSO:

Salesforce Object Query Language (SOQL) Reference

Bulk API 2.0 End-of-Life

Salesforce is committed to supporting each API version for a minimum of three years from the date of first release. In order to mature and improve the quality and performance of the API, versions that are more than three years old might cease to be supported.

When an API version is to be deprecated, advance notice is given at least one year before support ends. Salesforce will directly notify customers using API versions planned for deprecation.

Bulk API 2.0 Older Documentation

Find versions of the Bulk API 2.0 documentation released prior to Summer '21 (API version 52.0).

Document	Location
Spring '21 (API version 51.0)	https://resources.docs.salesforce.com/230/latest/en-us/sfdc/pdf/api_bulk_v2.pdf

Document	Location
Winter '21 (API version 50.0)	https://resources.docs.salesforce.com/228/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Summer '20 (API version 49.0)	https://resources.docs.salesforce.com/226/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Spring '20 (API version 48.0)	https://resources.docs.salesforce.com/224/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Winter '20 (API version 47.0)	https://resources.docs.salesforce.com/222/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Summer '19 (API version 46.0)	https://resources.docs.salesforce.com/220/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Spring '19 (API version 45.0)	https://resources.docs.salesforce.com/218/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Winter '19 (API version 44.0)	https://resources.docs.salesforce.com/216/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Summer '18 (API version 43.0)	https://resources.docs.salesforce.com/214/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Spring '18 (API version 42.0)	https://resources.docs.salesforce.com/212/latest/en-us/sfdc/pdf/api_bulk_v2.pdf
Winter '18 (API version 41.0)	https://resources.docs.salesforce.com/210/latest/en-us/sfdc/pdf/api_bulk_v2.pdf

CHAPTER 4 Bulk API

In this chapter ...

Bulk API is the predecessor to the current Bulk API 2.0. Although Bulk API gives you more fine-grained control over the specifics of jobs and batches, its work-flow is more complex than Bulk API 2.0. If the feature set and limits are a unique match to your project requirements, use Bulk API.

How Bulk API Works

You process a set of records by creating a job that contains one or more batches.

The job specifies which object is being processed and what type of operation is being used. A batch is a set of records sent to the server in an HTTP POST request. Each batch is processed independently by the server, not necessarily in the order it's received. Batches can be processed in parallel. It's up to the client to decide how to divide the entire data set into a suitable number of batches.

A job is represented by the JobInfo resource. This resource is used to create a new job, get status for an existing job, and change status for a job. A batch is created by submitting a CSV, XML, or JSON representation of a set of records and any references to binary attachments in an HTTP POST request. When created, the status of a batch is represented by a BatchInfo resource. When a batch is complete, the result for each record is available in a result set resource.

Processing data typically consists of these steps.

1. Create a new job that specifies the object and action.
2. Send data to the server in a number of batches.
3. After all data has been submitted, close the job. When closed, no more batches can be sent as part of the job.
4. Check status of all batches at a reasonable interval. Each status check returns the state of each batch.
5. When all batches have completed or failed, retrieve the result for each batch.
6. Match the result sets with the original data set to determine which records failed and succeeded, and take appropriate action.

You can abort the job at any point in this process. Aborting a job has the effect of preventing any unprocessed batches from being processed. It doesn't undo the effects of batches already processed.

Limits

Note the following limits specific to Bulk API.

API usage limits
Bulk API use is subject to the standard API usage limits. Each HTTP request counts as one call for the purposes of calculating usage limits.

Batch content
Each batch must contain exactly one CSV, XML, or JSON file containing records for a single object, or the batch isn't processed and `stateMessage` is updated. Use the Enterprise WSDL for the correct format for object records.

Batch Allocation
You can submit up to 15,000 batches per rolling 24-hour period. You can't create batches associated with a job that is more than 24 hours old. If a batch is submitted to a closed job, the batch isn't created, but it still counts against the batch allocation as a submitted batch.

Batch lifespan
Batches and jobs that are older than 7 days are removed from the queue if they're in a terminal state (completed or failed). The 7 days are measured from the youngest batch associated with a job, or the age of the job if there are no batches. You can't create batches associated with a job that is more than 24 hours old.

Batch size

- Batches for data loads can consist of a single CSV, XML, or JSON file that is no larger than 10 MB.
- A batch must contain some content or an error occurs.

For binary content limits, see General Limits on page 7.

Batch processing time

Batches are processed in chunks. The chunk size depends on the API version. In API version 20.0 and earlier, the chunk size is 100 records. In API version 21.0 and later, the chunk size is 200 records. There's a 5-minute limit for processing each chunk. Also, if it takes longer than 10 minutes to process a whole batch, the Bulk API places the remainder of the batch back in the queue for later processing. If the Bulk API continues to exceed the 10-minute limit on subsequent attempts, the batch is placed back in the queue and reprocessed up to 10 times before the batch is permanently marked as failed.

Even if the batch failed, some records could have completed successfully. If you get a timeout error when processing a batch, split your batch into smaller batches, and try again.

Note: Bulk queries have an extra 2-minute limit for processing the query, which is separate from the batch processing limit.

Compression

The only valid compression value is `gzip`. Compression is optional, but recommended. Compression doesn't affect the character limits defined in Batch size.

Job abort

Any user with correct permission can abort a job. Only the user who created a job can close it.

Job close

Only the user who created a job can close it. Any user with correct permission can abort a job.

Job content

Each job can specify one operation and one object. Batches associated with this job contain records of one object. Optionally, you can specify serial processing mode, which is used only when previously submitted asynchronous jobs have accidentally produced contention because of locks. Use only when advised by Salesforce.

Job external ID

You can't edit the value of an external ID field in JobInfo. When specifying an external ID, the operation must be upsert. If you try to use it with create or update, an error is generated.

Job lifespan

Batches and jobs that are older than 7 days are removed from the queue if they're in a terminal state (completed or failed). The 7 days are measured from the youngest batch associated with a job, or the age of the job if there are no batches. You can't create batches associated with a job that is more than 24 hours old.

Job open time

The maximum time that a job can remain open is 24 hours. The Bulk API doesn't support clients that, for example, post one batch every hour for many hours.

Job status in job history

The job status and batch results sets for completed jobs are available for 7 days, after which the data is deleted permanently.

Job status change

When you submit a POST body with a change in job status, you can only specify the `status` field value. If `operation` or `entity` field values are specified, an error occurs.

Portal users

Regardless of whether the API Enabled profile permission is granted, portal users (Customer Portal, Self-Service portal, and Partner Portal) can't access Bulk API.

SOQL

Bulk API doesn't support queries with any of the following:

- GROUP BY, OFFSET, or TYPEOF clauses
- Aggregate functions such as `COUNT()`
- Date functions in GROUP BY clauses (date functions in WHERE clauses are supported)

- Compound address fields or compound geolocations fields

Work with Jobs

You process a set of records by creating a job that contains one or more batches. The job specifies which object is being processed and what type of operation is being used. Ingest jobs are defined by CSV, XML, JSON, or binary attachments. Query jobs are defined by a SOQL statement.

A job is represented by the JobInfo resource. This resource is used to create a new job, get status for an existing job, and change status for a job.

Note: Salesforce provides an additional API, Bulk API 2.0, which uses the REST API framework to provide similar capabilities to Bulk API. Bulk API 2.0 simplifies the job creation and monitoring process. For more information on Bulk API 2.0, see the Bulk API 2.0 Developer Guide.

1. Create a Job
   Create a job by sending a POST request to this URI. The request body identifies the type of object processed in all associated batches.
2. Monitor a Job
   You can monitor a Bulk API job in Salesforce. The monitoring page tracks jobs and batches created by any client application, including Data Loader or any client application that you write.
3. Close a Job
   Close a job by sending a POST request to this URI. The request URI identifies the job to close. When a job is closed, no more batches can be added.
4. Get Job Details
   Get all details for an existing job by sending a GET request to this URI.
5. Abort a Job
   Abort an existing job by sending a POST request to this URI. The request URI identifies the job to abort. When a job is aborted, no more records are processed. If changes to data have been committed, they aren't rolled back.
6. Job and Batch Lifespan
   All jobs and batches older than 7 days are deleted.

Create a Job

Create a job by sending a POST request to this URI. The request body identifies the type of object processed in all associated batches.

URI

`/services/async/`***`APIversion`***`/job`

Example XML request body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <operation>insert</operation>
 <object>Account</object>
 <contentType>CSV</contentType>
</jobInfo>
```

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D00000004SkLIAU</id>
  <operation>insert</operation>
  <object>Account</object>
  <createdById>005D0000001b0fFIAQ</createdById>
  <createdDate>2015-12-15T21:41:45.000Z</createdDate>
  <systemModstamp>2015-12-15T21:41:45.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

Example JSON request body

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

Example JSON response body

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 0,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005D0000001b0fFIAQ",
   "createdDate" : "2015-12-15T20:45:25.000+0000",
   "id" : "750D00000004SkGIAU",
   "numberBatchesCompleted" : 0,
   "numberBatchesFailed" : 0,
   "numberBatchesInProgress" : 0,
   "numberBatchesQueued" : 0,
   "numberBatchesTotal" : 0,
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 0,
   "numberRetries" : 0,
   "object" : "Account",
   "operation" : "insert",
   "state" : "Open",
   "systemModstamp" : "2015-12-15T20:45:25.000+0000",
   "totalProcessingTime" : 0
}
```

In these samples, the `contentType` field indicates that the batches associated with the job are in CSV format. For alternative options, such as XML or JSON, see JobInfo.

Warning: The operation value must match that shown in JobInfo. For example, you get an error if you use INSERT instead of insert.

SEE ALSO:

- Create a Job for Batches with Binary Attachments
- Get Job Details
- Close a Job
- Abort a Job
- Add a Batch to a Job
- Job and Batch Lifespan
- Limits
- About URIs
- JobInfo
- Quick Start: Bulk API 2.0

Monitor a Job

You can monitor a Bulk API job in Salesforce. The monitoring page tracks jobs and batches created by any client application, including Data Loader or any client application that you write.

To track the status of bulk data load jobs that are in progress or recently completed, enter *`Bulk Data Load Jobs`* in the `Quick Find` box, then select **Bulk Data Load Jobs**. This page allows you to monitor the progress of current jobs and the results of recent jobs.

For more information, see Manage Bulk Data Load Jobs in the Salesforce online help.

SEE ALSO:

- Create a Job
- Get Job Details
- Close a Job
- Abort a Job
- Add a Batch to a Job
- Job and Batch Lifespan
- Limits
- Data Loader Guide

Close a Job

Close a job by sending a POST request to this URI. The request URI identifies the job to close. When a job is closed, no more batches can be added.

URI

/services/async/***APIversion***/job/*jobId*

Example XML request body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <state>Closed</state>
</jobInfo>
```

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>750D0000000002lIAA</id>
 <operation>insert</operation>
 <object>Account</object>
 <createdById>005D0000001ALVFIA4</createdById>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
 <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
 <state>Closed</state>
 <concurrencyMode>Parallel</concurrencyMode>
 <contentType>XML</contentType>
 <numberBatchesQueued>0</numberBatchesQueued>
 <numberBatchesInProgress>0</numberBatchesInProgress>
 <numberBatchesCompleted>1</numberBatchesCompleted>
 <numberBatchesFailed>0</numberBatchesFailed>
 <numberBatchesTotal>1</numberBatchesTotal>
 <numberRecordsProcessed>2</numberRecordsProcessed>
 <numberRetries>0</numberRetries>
 <apiVersion>36.0</apiVersion>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>3647</totalProcessingTime>
 <apiActiveProcessingTime>2136</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

Example JSON request body

```
{
   "state" : "Closed"
}
```

Example JSON response body

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 5059,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005xx000001SyhGAAS",
   "createdDate" : "2015-11-19T01:45:03.000+0000",
   "id" : "750xx000000000GAAQ",
   "numberBatchesCompleted" : 10,
   "numberBatchesFailed" : 0,
   "numberBatchesInProgress" : 0,
   "numberBatchesQueued" : 0,
   "numberBatchesTotal" : 10,
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 100,
   "numberRetries" : 0,
   "object" : "Account",
   "operation" : "insert",
   "state" : "Closed",
   "systemModstamp" : "2015-11-19T01:45:03.000+0000",
   "totalProcessingTime" : 5759
}
```

SEE ALSO:

- Create a Job
- Monitor a Job
- Get Job Details
- Abort a Job
- Job and Batch Lifespan
- Limits
- About URIs
- JobInfo
- Quick Start: Bulk API 2.0

Get Job Details

Get all details for an existing job by sending a GET request to this URI.

URI

/services/async/***APIversion***/job/*jobId*

Example request body

No request body is allowed.

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D00000004SkLIAU</id>
  <operation>insert</operation>
  <object>Account</object>
  <createdById>005D0000001b0fFIAQ</createdById>
  <createdDate>2015-12-15T21:41:45.000Z</createdDate>
  <systemModstamp>2015-12-15T21:41:45.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

Example JSON response body

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 0,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005D0000001b0fFIAQ",
   "createdDate" : "2015-12-15T20:45:25.000+0000",
   "id" : "750D00000004SkGIAU",
   "numberBatchesCompleted" : 0,
   "numberBatchesFailed" : 0,
   "numberBatchesInProgress" : 0,
   "numberBatchesQueued" : 0,
   "numberBatchesTotal" : 0,
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 0,
   "numberRetries" : 0,
   "object" : "Account",
   "operation" : "insert",
   "state" : "Open",
   "systemModstamp" : "2015-12-15T20:45:25.000+0000",
   "totalProcessingTime" : 0
}
```

SEE ALSO:

- Create a Job
- Monitor a Job
- Close a Job
- Abort a Job
- Add a Batch to a Job
- Job and Batch Lifespan
- Limits
- About URIs
- JobInfo
- Quick Start: Bulk API 2.0

Abort a Job

Abort an existing job by sending a POST request to this URI. The request URI identifies the job to abort. When a job is aborted, no more records are processed. If changes to data have been committed, they aren't rolled back.

URI

`/services/async/APIversion/job/jobId`

Example XML request body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <state>Aborted</state>
</jobInfo>
```

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>750D0000000002lIAA</id>
 <operation>insert</operation>
 <object>Account</object>
 <createdById>005D0000001ALVFIA4</createdById>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
 <systemModstamp>2009-04-14T18:16:00.000Z</systemModstamp>
 <state>Aborted</state>
 <concurrencyMode>Parallel</concurrencyMode>
 <contentType>XML</contentType>
 <numberBatchesQueued>0</numberBatchesQueued>
 <numberBatchesInProgress>0</numberBatchesInProgress>
 <numberBatchesCompleted>1</numberBatchesCompleted>
 <numberBatchesFailed>0</numberBatchesFailed>
 <numberBatchesTotal>1</numberBatchesTotal>
 <numberRecordsProcessed>2</numberRecordsProcessed>
 <numberRetries>0</numberRetries>
 <apiVersion>36.0</apiVersion>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>3647</totalProcessingTime>
 <apiActiveProcessingTime>2136</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

Example JSON request body

```
{
   "state" : "Aborted"
}
```

Example JSON response body

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 2166,
   "apiVersion" : 36.0,
   "concurrencyMode" : "Parallel",
   "contentType" : "JSON",
   "createdById" : "005D0000001b0fFIAQ",
   "createdDate" : "2015-12-15T21:54:29.000+0000",
   "id" : "750D00000004SkVIAU",
   "numberBatchesCompleted" : 2,
   "numberBatchesFailed" : 0,
   "numberBatchesInProgress" : 0,
   "numberBatchesQueued" : 0,
   "numberBatchesTotal" : 2,
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 2,
   "numberRetries" : 0,
   "object" : "Account",
   "operation" : "insert",
   "state" : "Aborted",
   "systemModstamp" : "2015-12-15T21:54:29.000+0000",
   "totalProcessingTime" : 2870
}
```

SEE ALSO:

Get Job Details

Create a Job

Monitor a Job

Close a Job

Job and Batch Lifespan

Limits

About URIs

JobInfo

Job and Batch Lifespan

All jobs and batches older than 7 days are deleted.

- It may take up to 24 hours for jobs and batches to be deleted once they're older than 7 days.
- If a job is more than 7 days old, but contains a batch that is less than 7 days old, then all the batches associated with that job, and the job itself, aren't deleted until the youngest batch is more than 7 days old.
- Jobs and batches are deleted regardless of status.
- After they're deleted, jobs and batches can't be retrieved from the platform.

For more information about limits, see Limits on page 78.

SEE ALSO:

Create a Job
Monitor a Job
Get Job Details
Close a Job
Abort a Job
Add a Batch to a Job
Limits
About URIs
JobInfo
Quick Start: Bulk API 2.0

Bulk API Ingest

With Bulk API, you can insert, update, or upsert large data sets into your Salesforce org. Prepare a CSV, XML, or JSON file representation of the data you want to upload, create a job, upload job data, and let Salesforce take care of the rest.

Plan Bulk Data Loads
Bulk API performance depends on the type of data that you're loading, as well as any workflow rules and triggers associated with the objects in your batches. It's useful to understand the factors that determine optimal loading time.

Install cURL
The Bulk API uses HTTP GET and HTTP POST methods to send and receive CSV, XML, and JSON content, so it's simple to build client applications using the tool or the language of your choice. This quick start uses a command-line tool called cURL to simplify sending and receiving HTTP requests and responses.

Walkthrough Sending HTTP Requests with cURL
With cURL now configured, you can start sending HTTP requests to the Bulk API. You send HTTP requests to a URI to perform operations with Bulk API.

Prepare Data Files
The Bulk API processes records in comma-separated values (CSV) files, XML files, or JSON files.

Load Binary Attachments
The Bulk API can load binary attachments, which can be Attachment objects or Salesforce CRM Content.

Request Basics

Here are some Bulk API request basics, including the format of URIs used to perform operations and details on how to authenticate requests using a session header.

Work with Batches

A batch is a set of records sent to the server in an HTTP POST request. Each batch is processed independently by the server, not necessarily in the order it's received.

Bulk API Reference

These are the supported resources for the Bulk API, as well as the details on errors and processing limits.

Plan Bulk Data Loads

Bulk API performance depends on the type of data that you're loading, as well as any workflow rules and triggers associated with the objects in your batches. It's useful to understand the factors that determine optimal loading time.

General Guidelines for Data Loads

For optimal processing time, consider these tips when planning your data loads. Always test your data loads in a sandbox organization first. Note that the processing times can be different in a production organization.

Use Compression for Responses

In API version 27.0 and later, Bulk API can compress response data which reduces network traffic and improves response time.

Configure Salesforce CORS Allowlist

Cross-Origin Resource Sharing (CORS) allows web browsers to request resources from other origins. For example, using CORS, the JavaScript for a web application at `https://www.example.com` can request a resource from `https://www.salesforce.com`. To allow access to supported Salesforce APIs, Apex REST resources, and Lightning Out from JavaScript code in a web browser, add the requesting origin to your Salesforce CORS allowlist. For Lightning apps that allow web browsers to make requests from their orgs, CORS allowlist prevents requests to Lightning apps unless the request comes from an approved URL.

General Guidelines for Data Loads

For optimal processing time, consider these tips when planning your data loads. Always test your data loads in a sandbox organization first. Note that the processing times can be different in a production organization.

Use Parallel Mode Whenever Possible

You get the most benefit from the Bulk API by processing batches in parallel, which is the default mode and enables faster loading of data. However, sometimes parallel processing can cause lock contention on records. The alternative is to process using serial mode. Don't process data in serial mode unless you know this would otherwise result in lock timeouts and you can't reorganize your batches to avoid the locks.

You set the processing mode at the job level. All batches in a job are processed in parallel or serial mode.

Organize Batches to Minimize Lock Contention

For example, when an AccountTeamMember record is created or updated, the account for this record is locked during the transaction. If you load many batches of AccountTeamMember records and they all contain references to the same account, they all try to lock the same account and it's likely that you'll experience a lock timeout. Sometimes, lock timeouts can be avoided by organizing data in batches. If you organize AccountTeamMember records by `AccountId` so that all records referencing the same account are in a single batch, you minimize the risk of lock contention by multiple batches.

The Bulk API doesn't generate an error immediately when encountering a lock. It waits a few seconds for its release and, if it doesn't happen, the record is marked as failed. If there are problems acquiring locks for more than 100 records in a batch, the Bulk API places the remainder of the batch back in the queue for later processing. When the Bulk API processes the batch again later, records marked as failed are not retried. To process these records, you must submit them again in a separate batch.

If the Bulk API continues to encounter problems processing a batch, it's placed back in the queue and reprocessed up to 10 times before the batch is permanently marked as failed. Even if the batch failed, some records could have completed successfully. If errors persist, create a separate job to process the data in serial mode, which ensures that only one batch is processed at a time.

Be Aware of Operations that Increase Lock Contention

These operations are likely to cause lock contention and necessitate using serial mode:

- Creating new users
- Updating ownership for records with private sharing
- Updating user roles
- Updating territory hierarchies

If you encounter errors related to these operations, create a separate job to process the data in serial mode.

Note: Because your data model is unique to your organization, Salesforce can't predict exactly when you might see lock contention problems.

Minimize Number of Fields

Processing time is faster if there are fewer fields loaded for each record. Foreign key, lookup relationship, and roll-up summary fields are more likely to increase processing time. It's not always possible to reduce the number of fields in your records, but if it is, loading times improve.

Minimize Number of Workflow Actions

Workflow actions increase processing time.

Minimize Number of Triggers

You can use parallel mode with objects that have associated triggers if the triggers don't cause side-effects that interfere with other parallel transactions. However, Salesforce doesn't recommend loading large batches for objects with complex triggers. Instead, rewrite the trigger logic as a batch Apex job that is executed after all the data has loaded.

Optimize Batch Size

Salesforce shares processing resources among all its customers. To ensure that each organization doesn't wait too long to process its batches, any batch that takes more than 10 minutes is suspended and returned to the queue for later processing. The best course of action is to submit batches that process in less than 10 minutes. For more information on monitoring timing for batch processing, see Monitor a Batch on page 115.

Adjust batch sizes based on processing times. Start with 5000 records and adjust the batch size based on processing time. If it takes more than 5 minutes to process a batch, it can be beneficial to reduce the batch size. If it takes a few seconds, increase the batch size. If you get a timeout error when processing a batch, split your batch into smaller batches, and try again. For more information, see Limits on page 78.

Note: For Bulk queries, the batch size isn't applied to the query result set or the retrieved data size. If your bulk query takes too long to process, filter your query statement to return less data.

Minimize Number of Batches in the Asynchronous Queue

Salesforce uses a queue-based framework to handle asynchronous processes from such sources as future and batch Apex, and Bulk API batches. This queue is used to balance request workload across organizations. If more than 2,000 unprocessed requests from a single organization are in the queue, any more requests from the same organization will be delayed while the queue handles requests from other organizations. Minimize the number of batches submitted at one time to ensure that your batches are not delayed in the queue.

Use Compression for Responses

In API version 27.0 and later, Bulk API can compress response data which reduces network traffic and improves response time.

Responses are compressed if the client makes a request using the `Accept-Encoding` header, with a value of `gzip`. Bulk API compresses the response in gzip format and sends the response to the client with a `Content-Encoding: gzip` response header. If a request is made using the `Accept-Encoding` header with a value other than `gzip`, the encoding type is ignored, and the response is not compressed.

As an example, if a Batch Results request is made with the `Accept-Encoding: gzip` header, the response looks something like:

```
HTTP/1.1 200 OK
Date: Tue, 09 Oct 2012 18:36:45 GMT
Content-Type: text/csv; charset=UTF-8
Content-Encoding: gzip
Transfer-Encoding: chunked

...compressed response body...
```

Bulk API follows the HTTP 1.1 standards for response compression. Most clients automatically support compressed responses. Visit `https://developer.salesforce.com/page/Tools` for more information on particular clients.

Configure Salesforce CORS Allowlist

EDITIONS

Available in: Salesforce Classic (not available in all orgs) and Lightning Experience

Available in: **Developer**, **Enterprise**, **Performance**, and **Unlimited** Editions

Available with API access enabled in: Professional Edition

USER PERMISSIONS

To create, read, update, and delete:

- Modify All Data

Cross-Origin Resource Sharing (CORS) allows web browsers to request resources from other origins. For example, using CORS, the JavaScript for a web application at `https://www.example.com` can request a resource from `https://www.salesforce.com`. To allow access to supported Salesforce APIs, Apex REST resources, and Lightning Out from JavaScript code in a web browser, add the requesting origin to your Salesforce CORS allowlist. For Lightning apps that allow web browsers to make requests from their orgs, CORS allowlist prevents requests to Lightning apps unless the request comes from an approved URL.

These Salesforce technologies support CORS.

- Apex REST
- Bulk API
- Bulk API 2.0
- Connect REST API
- Lightning Out
- REST API
- Salesforce IoT REST API
- CRM Analytics REST API
- User Interface API

Add an origin serving the request code to the CORS allowlist. If a browser that supports CORS makes a request to an origin in the allowlist, Salesforce returns the origin in the `Access-Control-Allow-Origin` HTTP header along with any additional CORS HTTP headers. If the origin isn't included in the allowlist, Salesforce returns HTTP status code 403.

1. From Setup, in the Quick Find box, enter *CORS*, and then select **CORS**.
2. Select **New**.
3. Enter a resource in Origin URL Pattern.

Tip: The origin URL pattern doesn't always match the URL that appears in your browser's address bar.

4. Save your changes.

The origin URL pattern must include the HTTPS protocol (unless you're using your localhost) and a domain name. It can also include a port. The wildcard character (*) is supported and must be in front of a second-level domain name. For example, `https://*.example.com` adds all subdomains of `example.com` to the allowlist.

The origin URL pattern can be an IP address. But an IP address and a domain that resolve to the same address aren't the same origin, and you must add them to the CORS allowlist as separate entries.

Google Chrome™ and Mozilla® Firefox® browser extensions are allowed as resources in API version 53 (Winter '22) or later . Chrome extensions must use the prefix `chrome-extension://` and 32 characters without digits or capital letters, for example `chrome-extension://abdkkegmcbiomijcbdaodaflgehfffed`. Firefox extensions must use the prefix `moz-extension://` and an 8-4-4-4-12 format of small alphanumeric characters, for example `moz-extension://1234ab56-78c9-1df2-3efg-4567891hi1j2`.

You can get a successful response when requesting a REST resource in a CORS preflight test, but receive an unsuccessful response to the actual request. This discrepancy can occur when the resource is deleted after the preflight test and before the request is made. It can also occur if the resource doesn't exist. A CORS preflight confirms if resources can be passed between servers, but doesn't check if a specific resource exists or not. CORS preflight requests are typically issued automatically by a browser.

Note: To access certain OAuth endpoints with CORS, other requirements apply. See Enable CORS for OAuth Endpoints.

Install cURL

The Bulk API uses HTTP GET and HTTP POST methods to send and receive CSV, XML, and JSON content, so it's simple to build client applications using the tool or the language of your choice. This quick start uses a command-line tool called cURL to simplify sending and receiving HTTP requests and responses.

cURL is pre-installed on many Linux and Mac systems. Windows users can download a version at `curl.haxx.se/`. When using HTTPS on Windows, ensure that your system meets the cURL requirements for SSL.

Note: cURL is an open-source tool and isn't supported by Salesforce.

Escaping the Session ID or Using Single Quotes on Mac and Linux Systems

When running the cURL examples, you can get an error on Mac and Linux systems due to the presence of the exclamation mark special character in the session ID argument. To avoid getting this error, use one of these solutions:

- Escape the exclamation mark (!) special character in the session ID by inserting a backslash before it (\!) when the session ID is enclosed within double quotes. For example, the session ID string in this cURL command has the exclamation mark (!) escaped:

```
curl https://MyDomainName.my.salesforce.com/services/async/55.0/job
-H "X-SFDC-Session:
00D50000000IehZ\!AQcAQH0dMHZfz972Szmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1E6
LYUfiDUkWe6H34r1AAwOR8B8fLEz6n04NPGRrq0FM"
```

- Enclose the session ID within single quotes. For example:

```
curl https://MyDomainName.my.salesforce.com/services/async/55.0/job
-H 'X-SFDC-Session: sessionID'
```

Walkthrough Sending HTTP Requests with cURL

With cURL now configured, you can start sending HTTP requests to the Bulk API. You send HTTP requests to a URI to perform operations with Bulk API.

The URI where you send HTTP requests has this format:

https://*`Web_Services_SOAP_endpoint_hostame`*`/services/async/`*`APIversion`*`/`*`Resource_address`*

The part after the API version (*`Resource_address`*) varies depending on the job or batch being processed.

The easiest way to start using the Bulk API is to enable it for processing records in Data Loader using CSV files. If you use Data Loader, you don't need to craft your own HTTP requests or write your own client application. For an example of writing a client application using Java, see Sample Client Application Using Java on page 156.

Step 1: Log In Using the SOAP API
The Bulk API doesn't provide a login operation, so you must use SOAP API to log in.

Step 2: Create a Job
Before you can load data, you first create a job. The job specifies the type of object, such as Contact, that you're loading and the operation that you're performing, such as query, queryAll, insert, update, upsert, or delete. A job also grants you some control over the data load process. For example, you can abort a job that is in progress.

Step 3: Add a Batch to the Job
After creating the job, you're ready to create a batch of contact records. You send data in batches in separate HTTP POST requests. The URI for each request is similar to the one you used when creating the job, but you append ***`jobId`***`/batch` to the URI.

Step 4: Close the Job
When you're finished submitting batches to Salesforce, close the job. Closing the job tells Salesforce that you're done submitting batches for the job, which allows the monitoring page in Salesforce to return more meaningful statistics on the progress of the job.

Step 5: Check Batch Status
You can check the status of an individual batch by running this cURL command.

Step 6: Retrieve Batch Results
When a batch is `Completed`, you must retrieve the batch result to see the status of individual records.

SEE ALSO:

About URIs

Data Loader Guide

Step 1: Log In Using the SOAP API

The Bulk API doesn't provide a login operation, so you must use SOAP API to log in.

To establish a session, you need to use SOAP API's `login()` function as described in the *SOAP API Developer Guide*:

- Step 4: Walk Through the Sample Code.
- login().

The `login()` function returns an XML response that includes `<sessionId>` and `<serverUrl>` elements. Note the values of the `<sessionId>` element and the first part of the host name (instance), such as ***`yourInstance`***`-api`, from the `<serverUrl>` element. Use these values in subsequent requests to the Bulk API.

SEE ALSO:

Set a Session Header

SOAP API Developer Guide

Step 2: Create a Job

Before you can load data, you first create a job. The job specifies the type of object, such as Contact, that you're loading and the operation that you're performing, such as query, queryAll, insert, update, upsert, or delete. A job also grants you some control over the data load process. For example, you can abort a job that is in progress.

1. Create a text file called `job.txt` containing this text:

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
    <operation>insert</operation>
    <object>Contact</object>
    <contentType>CSV</contentType>
</jobInfo>
```

Warning: The operation value must match that shown here. For example, you get an error if you use `INSERT` instead of `insert`.

2. Using a command-line window, execute the following cURL command:

```
curl https://instance.salesforce.com/services/async/55.0/job -H "X-SFDC-Session:
sessionId" -H "Content-Type: application/xml; charset=UTF-8" -d @job.txt
```

`instance` is the portion of the `<serverUrl>` element and *`sessionId`* is the `<sessionId>` element that you noted in the login response.

Note: When running cURL examples, you can get an error on Mac and Linux systems due to the presence of the exclamation mark special character in the session ID argument. To avoid getting this error, either escape the exclamation mark by inserting a backslash before it (\!) or enclose the session ID within single quotes.

Salesforce returns an XML response with data such as this:

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x0000000005LAAQ</id>
  <operation>insert</operation>
  <object>Contact</object>
  <createdById>005x0000000wPWdAAM</createdById>
  <createdDate>2009-09-01T16:42:46.000Z</createdDate>
  <systemModstamp>2009-09-01T16:42:46.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>55.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

3. Note the value of the job ID returned in the `<id>` element. Use this ID in subsequent operations.

SEE ALSO:

Create a Job

Step 3: Add a Batch to the Job

After creating the job, you're ready to create a batch of contact records. You send data in batches in separate HTTP POST requests. The URI for each request is similar to the one you used when creating the job, but you append ***`jobId`***`/batch` to the URI.

Format the data as CSV, XML, or JSON if you're not including binary attachments. For information about binary attachments, see Load Binary Attachments on page 108. For information about batch size limitations, see Batch size and limits on page 78.

This example shows CSV as this is the recommended format. It's your responsibility to divide up your data set in batches that fit within the limits. In this example, we keep it very simple with just a few records.

To add a batch to a job:

1. Create a CSV file named `data.csv` with these two records.

```
FirstName,LastName,Department,Birthdate,Description
Tom,Jones,Marketing,1940-06-07Z,"Self-described as ""the top"" branding guru on the West
 Coast"
Ian,Dury,R&D,,"World-renowned expert in fuzzy logic design.
Influential in technology purchases."
```

The value for the `Description` field in the last row spans multiple lines, so it's wrapped in double quotes.

2. Using a command-line window, execute this cURL command.

```
curl https://instance.salesforce.com/services/async/55.0/job/jobId/batch -H
"X-SFDC-Session: sessionId" -H "Content-Type: text/csv; charset=UTF-8" --data-binary
@data.csv
```

`instance` is the portion of the `<serverUrl>` element and *`sessionId`* is the `<sessionId>` element that you noted in the login response. *`jobId`* is the job ID that was returned when you created the job.

Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>751x00000000079AAA</id>
 <jobId>750x0000000005LAAQ</jobId>
```

```
  <state>Queued</state>
  <createdDate>2009-09-01T17:44:45.000Z</createdDate>
  <systemModstamp>2009-09-01T17:44:45.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Salesforce doesn't parse the CSV content or otherwise validate the batch until later. The response only acknowledges that the batch was received.

3. Note the value of the batch ID returned in the `<id>` element. You can use this batch ID later to check the status of the batch.

SEE ALSO:

Prepare CSV Files

Add a Batch to a Job

Limits

Step 4: Close the Job

When you're finished submitting batches to Salesforce, close the job. Closing the job tells Salesforce that you're done submitting batches for the job, which allows the monitoring page in Salesforce to return more meaningful statistics on the progress of the job.

1. Create a text file called `close_job.txt` containing the following text:

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <state>Closed</state>
</jobInfo>
```

2. Using a command-line window, execute the following cURL command:

```
curl https://instance.salesforce.com/services/async/55.0/job/jobId -H "X-SFDC-Session:
sessionId" -H "Content-Type: application/xml; charset=UTF-8" -d @close_job.txt
```

`instance` is the portion of the `<serverUrl>` element and *`sessionId`* is the `<sessionId>` element that you noted in the login response. *`jobId`* is the job ID that was returned when you created the job.

This cURL command updates the job resource state from `Open` to `Closed`.

SEE ALSO:

Close a Job

Step 5: Check Batch Status

You can check the status of an individual batch by running this cURL command.

```
curl https://instance.salesforce.com/services/async/55.0/job/jobId/batch/batchId -H
"X-SFDC-Session: sessionId"
```

instance is the portion of the `<serverUrl>` element and *sessionId* is the `<sessionId>` element that you noted in the login response. *jobId* is the job ID that was returned when you created the job. *batchId* is the batch ID that was returned when you added a batch to the job.

Salesforce returns an XML response with data such as this:

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x00000000079AAA</id>
  <jobId>750x0000000005LAAQ</jobId>
  <state>Completed</state>
  <createdDate>2009-09-01T17:44:45.000Z</createdDate>
  <systemModstamp>2009-09-01T17:44:45.000Z</systemModstamp>
  <numberRecordsProcessed>2</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>5820</totalProcessingTime>
  <apiActiveProcessingTime>2166</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

If Salesforce can't read the batch content or if the batch contains errors, such as invalid field names in the CSV header row, the batch state is `Failed`. When the batch state is `Completed`, all records in the batch have been processed. However, individual records could have failed. To see the status of individual records, retrieve the batch result.

Checking the status of each individual batch isn't necessary. You can check the status for all batches that are part of the job by running this cURL command:

```
curl https://instance.salesforce.com/services/async/55.0/job/jobId/batch -H
"X-SFDC-Session: sessionId"
```

SEE ALSO:

Get Information for a Batch

Get Information for All Batches in a Job

Interpret Batch State

Step 6: Retrieve Batch Results

When a batch is `Completed`, you must retrieve the batch result to see the status of individual records.

Retrieve the results for an individual batch by running the following cURL command:

```
curl https://instance.salesforce.com/services/async/55.0/job/jobId/batch/batchId/result
-H "X-SFDC-Session: sessionId"
```

instance is the portion of the `<serverUrl>` element and *sessionId* is the `<sessionId>` element that you noted in the login response. *jobId* is the job ID that was returned when you created the job. *batchId* is the batch ID that was returned when you added a batch to the job.

Salesforce returns a response with data such as this:

```
"Id","Success","Created","Error"
"003x0000004ouM4AAI","true","true",""
"003x0000004ouM5AAI","true","true",""
```

The response body is a CSV file with a row for each row in the batch request. If a record was created, the ID is contained in the row. If a record was updated, the value in the `Created` column is false. If a record failed, the `Error` column contains an error message.

SEE ALSO:

Get Batch Results

Handle Failed Records in Batches

Prepare Data Files

The Bulk API processes records in comma-separated values (CSV) files, XML files, or JSON files.

Note: For best performance, Salesforce recommends the following order of preference for data files: CSV, JSON, XML.

For information about loading records containing binary attachments, see Load Binary Attachments on page 108.

For information about loading data from third-party sources, see Map Data Fields on page 169.

Find Field Names
After you set up your client, you can build client applications that use the Bulk API. Use the following sample to create a client application. Each section steps through part of the code. The complete sample is included at the end.

Valid Date Format in Records
Specify the right format for `dateTime` and `date` fields.

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. A record consists of a series of fields that are delimited by commas. A CSV file can contain multiple records and constitutes a batch.

Prepare XML and JSON Files
The Bulk API processes records in XML, JSON, or CSV files. An XML or JSON file can contain multiple records and constitutes a batch. A record in an XML file is defined in an `sObjects` tag.

SEE ALSO:

Data Loader Guide

Find Field Names

After you set up your client, you can build client applications that use the Bulk API. Use the following sample to create a client application. Each section steps through part of the code. The complete sample is included at the end.

You can:

- Use the `describeSObjects()` call in the *SOAP API Developer Guide*, or the `sObject Describe` resource in the *REST API Developer Guide*.
- Use Salesforce Setup.
- Look up the object in *Object Reference*, which lists the field names, types, and descriptions by object.

Use Salesforce Setup to Find Field names

To find an object’s field name in Salesforce Setup:

1. From **Setup**, in the **Quick Find** box, enter *`Object Manager`*. Click **Object Manager**.
2. Click on the object in the list.
3. From the object's management settings, click on **Fields & Relationships**.
4. Click the field under `Field Label` to find the field name.

For a standard field, use the `Field Name` value as the field column header in your CSV file.

For a custom field, use the `API Name` value as the field column header in a CSV file or the field name identifier in an XML or JSON file. (To find the API Name, click the field name.)

SEE ALSO:

Salesforce Help: Find Object Management Settings

Valid Date Format in Records

Specify the right format for `dateTime` and `date` fields.

dateTime

Use the *`yyyy-MM-ddTHH:mm:ss.SSS+/-HH:mm`* 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)
- `+/-HH:mm` 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.

date

Use the *`yyyy-MM-dd`* format to specify `date` fields.

Note: Specifying an offset for `date` is not supported.

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. A record consists of a series of fields that are delimited by commas. A CSV file can contain multiple records and constitutes a batch.

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

Note the following when processing CSV files with the Bulk API:

- The Bulk API doesn't support any delimiter except for a comma.
- The Bulk API is optimized for processing large sets of data and has a strict format for CSV files. See Valid CSV Record Rows on page 101. The easiest way to process CSV files is to enable Bulk API for Data Loader.
- 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.

Tip: To import data from CSV files that don't meet these rules, map the data fields in the CSV file to Salesforce data fields. See Map Data Fields on page 169.

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 API to get the `relationshipName` property value for a field.

Valid CSV Record Rows

The Bulk API uses a strict format for field values to optimize processing for large sets of data.

Sample CSV File

The CSV sample includes two records for the Contact object. Each record contains six fields. You can include any field for an object that you're processing. If you use a CSV file to update existing accounts, fields that aren't defined in the CSV file are ignored during the update. Include all required fields when you create a record.

SEE ALSO:

Data Loader Guide

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 API to get the `relationshipName` property value for a field.

Some objects also have relationships to themselves. For example, the `ReportsTo` 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 `ReportsTo` field for a contact. The `ReportsTo` portion of the column header is the `relationshipName` property value for the `ReportsTo` 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. Because 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.

Valid CSV Record Rows

The Bulk API uses a strict format for field values to optimize processing for large sets of data.

Note the following when generating CSV files that contain Salesforce records:

- The delimiter for field values in a row must be a comma.
- If a field value contains a comma, a new line, or a double quote, the field value must be contained within double quotes: for example, `"Director of Operations, Western Region"`.
- If a field value contains a double quote, the double quote must be escaped by preceding it with another double quote: for example, `"This is the ""gold"" standard"`.

- Field values aren't trimmed. A space before or after a delimiting comma 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.

Sample CSV File

The CSV sample includes two records for the Contact object. Each record contains six fields. You can include any field for an object that you're processing. If you use a CSV file to update existing accounts, fields that aren't defined in the CSV file are ignored during the update. Include all required fields when you create a record.

This sample sets a relationship between the records that you're processing and existing contact records with the email addresses `buyer@salesforcesample.com` and `cto@salesforcesample.com`. If you try to insert the records in this sample into an org that doesn't contain contacts with those email addresses, the insertion fails.

To use this sample CSV for testing purposes, either remove `ReportsTo.Email` and its associated values or pre-insert contacts in your org that have these email addresses.

```
FirstName,LastName,Title,ReportsTo.Email,Birthdate,Description
Tom,Jones,Senior Director,buyer@salesforcesample.com,1940-06-07Z,"Self-described as ""the
 top"" branding guru on the West Coast"
Ian,Dury,Chief Imagineer,cto@salesforcesample.com,,"World-renowned expert in fuzzy logic
design.
Influential in technology purchases."
```

The `Description` field for the last record includes a line break, which is why the field value is enclosed in double quotes.

SEE ALSO:

Sample XML File

Sample JSON File

Data Loader Guide

Prepare XML and JSON Files

The Bulk API processes records in XML, JSON, or CSV files. An XML or JSON file can contain multiple records and constitutes a batch. A record in an XML file is defined in an `sObjects` tag.

All records in an XML or JSON file must be for the same object. You specify the object in the job associated with the batch. All batches associated with a job must contain records for the same object.

When processing XML or JSON files with the Bulk API:

- 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, fields not defined in the file are ignored during the update.
- Files must be in UTF-8 format.

Relationship Fields in Records

Many objects in Salesforce are related to other objects. For example, Account is a parent of Contact. Some objects also have relationships to themselves. For example, the `ReportsTo` field for a contact is a reference to another contact.

Valid XML and JSON Records

A batch request in an XML or JSON file contains records for one object type. Each batch in an XML file uses this format, with each `sObject` tag representing a record.

Sample XML File

This XML sample includes two records for the Account object. Each record contains three fields. You can include any field for an object that you're processing. If you use this file to update existing accounts, any fields that aren't defined in the XML file are ignored during the update. You must include all required fields when you create a record.

Sample JSON File

This JSON sample includes two records for the Account object. Each record contains three fields. You can include any field for an object that you're processing. If you use this file to update existing accounts, fields not defined in the JSON file are ignored during the update. You must include all required fields when you create a record.

Relationship Fields in Records

Many objects in Salesforce are related to other objects. For example, Account is a parent of Contact. Some objects also have relationships to themselves. For example, the `ReportsTo` field for a contact is a reference to another contact.

To add a reference to a related object for a field in a JSON or XML record, use the following syntax to represent the relationship. The *`RelationshipName`* is the relationship name of the field, and *`IndexedFieldName`* is the indexed field name that identifies the parent record.

JSON:

```
"RelationshipName" : { "IndexedFieldName" : "rwilliams@salesforcesample.com" }
```

XML:

```
<RelationshipName>
    <sObject>
        <IndexedFieldName>rwilliams@salesforcesample.com</IndexedFieldName>
    </sObject>
</RelationshipName>
```

Use the `describeSObjects()` call in the API to get the `relationshipName` property value for a field. Use an indexed field to uniquely identify the parent record for the relationship. A standard field is indexed if its `idLookup` property is set to `true`.

These samples include a contact record that includes the `ReportsTo` field, which is a reference to another contact. `ReportsTo` is the `relationshipName` property value for the `ReportsTo` field. In this case, the parent object for the `ReportsTo` field is also a contact, so we use the `Email` field to identify the parent record. The `idLookup` property value for the `Email` field is `true`. To see if there is a `idLookup` property for a field, go to the Field Properties column in the field table for each standard object.

JSON:

```
[{
  "FirstName" : "Ray",
  "LastName" : "Riordan",
  "ReportsTo" : { "Email" : "rwilliams@salesforcesample.com" }
}]
```

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <FirstName>Ray</FirstName>
```

```
        <LastName>Riordan</LastName>
        <ReportsTo>
          <sObject>
            <Email>rwilliams@salesforcesample.com</Email>
          </sObject>
        </ReportsTo>
    </sObject>
</sObjects>
```

When using relationships in JSON or XML records:

- 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.

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 using an indexed field. A custom field is indexed if its `External ID` field is selected.

For example, let's say a child object has a custom field with an `API Name` of `Mother_Of_Child__c` that points to a parent custom object. Let's assume that the parent object has a field with an `API Name` of `External_ID__c`. You can use the `Mother_Of_Child__r relationshipName` property to indicate that you're referencing a relationship to the parent object. Use the parent object's `External ID` field as a unique identifier for the `Mother Of Child` field. To use a relationship name, 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 JSON and XML files show usage of the relationship.

JSON:

```
[{
  "Name" : "CustomObject1",
  "Mother_Of_Child__r" : { "External_ID__c" : "123456" }
}]
```

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
    <sObject>
        <Name>CustomObject1</Name>
        <Mother_Of_Child__r>
          <sObject>
            <External_ID__c>123456</External_ID__c>
          </sObject>
          </Mother_Of_Child__r>
    </sObject>
</sObjects>
```

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 relationship field has an extra element to define the type of the parent object. The following JSON and XML samples show the syntax, where `RelationshipName` is the relationship name of the field, `ObjectTypeName` is the object type of the parent record, and `IndexedFieldName` is the indexed field name that uniquely identifies the parent record.

JSON:

```
"RelationshipName" : {
  "attributes" : {
    "type" : "ObjectTypeName" },
  "IndexedFieldName" : "rwilliams@salesforcesample.com"
}
```

XML:

```
<RelationshipName>
    <sObject>
        <type>ObjectTypeName</type>
        <IndexedFieldName>rwilliams@salesforcesample.com</IndexedFieldName>
    </sObject>
</RelationshipName>
```

These samples include 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.

JSON:

```
[{
  "Subject" : "Test Bulk API polymorphic reference field",
  "Priority" : "Normal",
  "Status" : "Not Started",
  "Who" : {
    "attributes" : {
      "type" : "Lead" },
      "Email" : "lead@salesforcesample.com" },
  "Owner" : { "Id" : "005D0000001AXYz" }
}]
```

XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <Subject>Test Bulk API polymorphic reference field</Subject>
      <Priority>Normal</Priority>
      <Status>Not Started</Status>
      <Who>
        <sObject>
          <type>Lead</type>
          <Email>lead@salesforcesample.com</Email>
        </sObject>
      </Who>
      <Owner>
```

```
        <sObject>
          <Id>005D0000001AXYz</Id>
        </sObject>
      </Owner>
   </sObject>
</sObjects>
```

Warning: The `type` element is required only for a polymorphic field. If you omit this element for a polymorphic field or include it for a non-polymorphic field, you get an error.

Valid XML and JSON Records

A batch request in an XML or JSON file contains records for one object type. Each batch in an XML file uses this format, with each `sObject` tag representing a record.

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <field_name>field_value</field_name>
      ...
   </sObject>
   <sObject>
      <field_name>field_value</field_name>
      ...
   </sObject>
</sObjects>
```

Each batch in a JSON file uses this format with each JSON object representing a record.

```
[
   {
      "field_name" : "field_value"
      ...
   },
   {
      "field_name" : "field_value"
      ...
   }
]
```

Note: You must include the `type` field for a polymorphic field and exclude it for non-polymorphic fields in any batch. If you don't the batch fails. 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.

When generating records in XML or JSON files:

- Fields that aren't defined in the file for a record are ignored when you update records. To set a field value to `null` in XML, set the `xsi:nil` value for the field to `true`. For example, `<description xsi:nil="true"/>` sets the description field to `null`. Setting `xsi:nil` to `false` has no effect. If you define a field value and set `xsi:nil` to `false`, the value still gets assigned. To specify a null value in JSON, set the field value to `null`. For example, `"description" : null`.

- 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.

SEE ALSO:

Sample XML File

Sample JSON File

Sample XML File

This XML sample includes two records for the Account object. Each record contains three fields. You can include any field for an object that you're processing. If you use this file to update existing accounts, any fields that aren't defined in the XML file are ignored during the update. You must include all required fields when you create a record.

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <Name>Xytrex Co.</Name>
      <Description>Industrial Cleaning Supply Company</Description>
      <Account Number>ABC15797531</Account Number>
   </sObject>
   <sObject>
      <Name>Watson and Powell, Inc.</Name>
      <Description>Law firm. New York Headquarters</Description>
      <Account Number>ABC24689753</Account Number>
   </sObject>
</sObjects>
```

SEE ALSO:

Sample CSV File

Sample JSON File

Sample JSON File

This JSON sample includes two records for the Account object. Each record contains three fields. You can include any field for an object that you're processing. If you use this file to update existing accounts, fields not defined in the JSON file are ignored during the update. You must include all required fields when you create a record.

This JSON sample includes two records for the Account object. Each record contains three fields. You can include any field for an object that you're processing. If you use this file to update existing accounts, fields not defined in the JSON file are ignored during the update. You must include all required fields when you create a record.

```
[
   {
      "Name" : "Xytrex Co.",
      "Description" : "Industrial Cleaning Supply Company",
      "Account Number" : "ABC15797531"
   },
   {
      "Name" : "Watson and Powell, Inc.",
      "Description" : "Law firm. New York Headquarters",
      "Account Number" : "ABC24689753"
```

```
    }
]
```

SEE ALSO:

Sample CSV File

Sample XML File

Load Binary Attachments

The Bulk API can load binary attachments, which can be Attachment objects or Salesforce CRM Content.

Create a request.txt File

A batch is represented by a zip file, which contains a CSV, XML, or JSON file called `request.txt` that contains references to the binary attachments and the binary attachments themselves. This differs from CSV, XML, or JSON batch files that don't include binary attachments. These batch files don't need a zip or a `request.txt` file.

Create a Zip Batch File with Binary Attachments

You can create a zip batch file for submitting your binary attachments as a batch.

Create a Job for Batches with Binary Attachments

You can use cURL to create a job for batches containing Attachment records.

Create a Batch with Binary Attachments

After creating the job, you're ready to create a batch of Attachment records. You send data in batches in separate HTTP POST requests. In this example, you create and submit one batch.

Create a request.txt File

A batch is represented by a zip file, which contains a CSV, XML, or JSON file called `request.txt` that contains references to the binary attachments and the binary attachments themselves. This differs from CSV, XML, or JSON batch files that don't include binary attachments. These batch files don't need a zip or a `request.txt` file.

The `request.txt` file is contained in the base directory of the zip file. The binary attachments can also be in the base directory or they can be organized in optional subdirectories. The `request.txt` file is a manifest file for the attachments in the zip file and contains the data for each record that references a binary file.

Note: The batch data file is named `request.txt`, whether you're working with CSV, XML, or JSON data.

For the Attachment object, the notation for these fields is particularly important:

- The `Name` field is the file name of the binary attachment. The easiest way to get a unique name for each attachment in your batch is to use the relative path from the base directory to the binary attachment. For example, `attachment1.gif` or `subdir/attachment2.doc`.
- The `Body` is the relative path to the binary attachment, preceded with a `#` symbol. For example, `#attachment1.gif` or `#subdir/attachment2.doc`.
- The `ParentId` field identifies the parent record, such as an account or a case, for the attachment.

The batch file can also include other optional Attachment fields, such as `Description`. For more information, see Attachment.

Sample CSV `request.txt` File

This sample CSV file includes two Attachment records. The first record references an `attachment1.gif` binary file in the base directory of the zip file. The second record references an `attachment2.doc` binary file in the `subdir` subdirectory of the zip file. In this example, the `ParentId` field indicates that both attachments are associated with Account parent records. The *`Account Id`* variable should be replaced with the `Id` of the associated parent account.

```
Name,ParentId,Body
attachment1.gif,Account Id,#attachment1.gif
subdir/attachment2.doc,Account Id,#subdir/attachment2.doc
```

Sample XML `request.txt` File

This sample XML file includes the same two records as the previous CSV sample file.

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects
   xmlns="http://www.force.com/2009/06/asyncapi/dataload"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <sObject>
    <Name>attachment1.gif</Name>
    <ParentId>Account Id</ParentId>
    <Body>#attachment1.gif</Body>
  </sObject>
  <sObject>
    <Name>subdir/attachment2.doc</Name>
    <ParentId>Account Id</ParentId>
    <Body>#subdir/attachment2.doc</Body>
  </sObject>
</sObjects>
```

Sample JSON `request.txt` File

This sample JSON file includes the same two records as the previous examples.

```
[
  {
    "Name" : "attachment1.gif",
    "ParentId" : "Account Id",
    "Body" : "#attachment1.gif"
  },
  {
    "Name" : "subdir/attachment2.doc",
    "ParentId" : "Account Id",
    "Body" : "#subdir/attachment2.doc"
  }
]
```

Create a Zip Batch File with Binary Attachments

You can create a zip batch file for submitting your binary attachments as a batch.

1. Create a base directory that contains the binary attachments. Attachments can be organized in subdirectories.

2. Create the `request.txt` CSV, XML, or JSON file in the base directory. The `request.txt` file is a manifest file for the attachments in the zip file and contains the data for each record that references a binary file.
3. Create a zip file of the base directory and any subdirectories.

Create a Job for Batches with Binary Attachments

You can use cURL to create a job for batches containing Attachment records.

For more information, see Walkthrough Sending HTTP Requests with cURL on page 93.

1. Create a text file called `job.txt` containing this text.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
    <operation>insert</operation>
    <object>Attachment</object>
    <contentType>ZIP_CSV</contentType>
</jobInfo>
```

Note: The batches for this job contain data in CSV format, so the `contentType` field is set to `ZIP_CSV`. For XML or JSON batches, use `ZIP_XML` or `ZIP_JSON`, respectively.

2. Using a command-line window, execute this cURL command

```
curl https://instance.salesforce.com/services/async/55.0/job -H "X-SFDC-Session:
sessionId" -H "Content-Type: application/xml; charset=UTF-8" -d @job.txt
```

`instance` is the portion of the `<serverUrl>` element and *`sessionId`* is the `<sessionId>` element that you noted in the login response. For more information about logging in, see Step 1: Log In Using the SOAP API on page 93.

Salesforce returns an XML response with data such as the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D000000001SRIAY</id>
  <operation>insert</operation>
  <object>Attachment</object>
  <createdById>005D0000001B0VkIAK</createdById>
  <createdDate>2010-08-25T18:52:03.000Z</createdDate>
  <systemModstamp>2010-08-25T18:52:03.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>ZIP_CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>55.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

3. Note the value of the job ID returned in the `<id>` element. Use this ID in subsequent operations.

Create a Batch with Binary Attachments

After creating the job, you're ready to create a batch of Attachment records. You send data in batches in separate HTTP POST requests. In this example, you create and submit one batch.

To organize your data in different batches, see General Guidelines for Data Loads on page 89.

1. Create a zip batch file. For this example, the file is named `request.zip`.
2. Using a command-line window, execute this cURL command.

   `curl https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***`/batch -H "X-SFDC-Session: `***`sessionId`***`" -H "Content-Type:zip/csv" --data-binary @request.zip`

   `instance` is the portion of the `<serverUrl>` element and *`sessionId`* is the `<sessionId>` element that you noted in the login response. *`jobId`* is the job ID that was returned when you created the job.

   Note: The Content-type for the POST request is `zip/csv`. For XML or JSON batches, use `zip/xml` or `zip/json` instead.

   Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751D000000003uwIAA</id>
  <jobId>750D000000001TyIAI</jobId>
  <state>Queued</state>
  <createdDate>2010-08-25T21:29:55.000Z</createdDate>
  <systemModstamp>2010-08-25T21:29:55.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

   Salesforce doesn't parse the CSV content or otherwise validate the batch until later. The response only acknowledges that the batch was received.

3. Note the value of the batch ID returned in the `<id>` element. You can use this batch ID later to check the status of the batch.

For details on proceeding to close the associated job, check batch status, and retrieve batch results, see the Getting Started.

Request Basics

Here are some Bulk API request basics, including the format of URIs used to perform operations and details on how to authenticate requests using a session header.

About URIs
You send HTTP requests to a URI to perform operations with Bulk API.

Set a Session Header

All HTTP requests must contain a valid API session ID obtained with the SOAP API `login()` call. The session ID is returned in the SessionHeader.

About URIs

You send HTTP requests to a URI to perform operations with Bulk API.

The URI where you send HTTP requests has this format:

https://*Web_Services_SOAP_endpoint_hostame*/services/async/*APIversion*/*Resource_address*

Think of the part of the URI through the API version as a base URI that's used for all operations. The part after the API version (*Resource_address*) varies depending on the job or batch being processed. For example, if you're working with version 55.0 of Bulk API in a production org, your base URI would be `https://`***MyDomainName***`.my.salesforce.com/services/async/55.0`.

You can find the My Domain name and My Domain login URL for your org on the My Domain page in Setup. Or, to get the hostname of your My Domain login URL in Apex, use the `getMyDomainHostname()` method of the `System.DomainCreator` class.

SEE ALSO:

Work with Jobs

Work with Batches

Set a Session Header

All HTTP requests must contain a valid API session ID obtained with the SOAP API `login()` call. The session ID is returned in the SessionHeader.

This example shows how to specify the required information after you obtain it from the `login()` call.

```
POST /services/async/55.0/job/ HTTP/1.1
Content-Type: application/xml; charset=UTF-8
Accept: application/xml
User-Agent: Salesforce Web Service Connector For Java/1.0
X-SFDC-Session: sessionId
Host: MyDomainName.my.salesforce.com
Connection: keep-alive
Content-Length: 135

<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <operation>insert</operation>
 <object>Account</object>
</jobInfo>
```

SEE ALSO:

Quick Start: Bulk API 2.0

Sample Client Application Using Java

Work with Batches

A batch is a set of records sent to the server in an HTTP POST request. Each batch is processed independently by the server, not necessarily in the order it's received.

A batch is created by submitting a CSV, XML, or JSON representation of a set of records and any references to binary attachments in an HTTP POST request. When created, the status of a batch is represented by a BatchInfo resource. When a batch is complete, the result for each record is available in a result set resource.

Batches can be processed in parallel. It's up to the client to decide how to divide the entire data set into a suitable number of batches.

Adjust batch sizes based on processing times. Start with 5000 records and adjust the batch size based on processing time. If it takes more than 5 minutes to process a batch, it can be beneficial to reduce the batch size. If it takes a few seconds, increase the batch size. If you get a timeout error when processing a batch, split your batch into smaller batches, and try again.

Note: Salesforce provides an additional API, Bulk API 2.0, which uses the REST API framework to provide similar capabilities to Bulk API. Bulk API 2.0 removes the need for creating and monitoring batches, and it lets you load record data for a job directly. For more information on Bulk API 2.0, see the Bulk API 2.0 Developer Guide.

1. Add a Batch to a Job
   Add a new batch to a job by sending a POST request to this URI. The request body contains a list of records for processing.
2. Monitor a Batch
   You can monitor a Bulk API batch in Salesforce.
3. Get Information for a Batch
   Get information about an existing batch by sending a GET request to this URI.
4. Get Information for All Batches in a Job
   Get information about all batches in a job by sending a GET request to this URI.
5. Interpret Batch State
   This list gives you more details about the various states, also known as statuses, of a batch. The batch state informs you whether to proceed to get the results, or whether you must wait or fix errors related to your request.
6. Get a Batch Request
   Get a batch request by sending a GET request to the following URI. This is available in API version 19.0 and later.
7. Get Batch Results
   Get results of a batch that completed processing by sending a GET request to this URI. If the batch is a CSV file, the response is in CSV format. If the batch is an XML or JSON file, the response is in XML or JSON format, respectively.
8. Handle Failed Records in Batches
   A batch can have a `Completed` state even if some or all of the records failed. If a subset of records failed, the successful records aren't rolled back. Likewise, even if the batch has a `Failed` state or if a job is aborted, some records could have completed successfully.

Add a Batch to a Job

Add a new batch to a job by sending a POST request to this URI. The request body contains a list of records for processing.

URI

/services/async/***APIversion***/job/***jobid***/batch

Note: The API version in the URI for all batch operations must match the API version for the associated job.

Example XML request body

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 0 (batch 0)</name>
  </sObject>
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 1 (batch 0)</name>
  </sObject>
</sObjects>
```

In this sample, the batch data is in XML format because the `contentType` field of the associated job was set to `XML`. For alternative formats for batch data, such as CSV or JSON, see JobInfo on page 125.

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>751D0000000004rIAA</id>
 <jobId>750D0000000002lIAA</jobId>
 <state>Queued</state>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
 <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
 <numberRecordsProcessed>0</numberRecordsProcessed>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>0</totalProcessingTime>
 <apiActiveProcessingTime>0</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Example JSON request body

```
[
   {
      "Name":"[Bulk API] Account 0 (batch 0)",
      "description" : "Created from Bulk API"
   },
   {
      "Name":"[Bulk API] Account 1 (batch 0)",
      "description" : "Created from Bulk API"
   }
]
```

In this sample, the batch data is in JSON format because the `contentType` field of the associated job was set to `JSON`.

Example JSON response body

```
{
   "apexProcessingTime":0,
   "apiActiveProcessingTime":0,
   "createdDate":"2015-12-15T21:56:43.000+0000",
   "id":"751D00000004YGZIA2",
   "jobId":"750D00000004SkVIAU",
   "numberRecordsFailed":0,
   "numberRecordsProcessed":0,
   "state":"Queued",
   "systemModstamp":"2015-12-15T21:56:43.000+0000",
   "totalProcessingTime":0
}
```

Note: You can add batch jobs using non-Bulk API-compliant CSV files. See Map Data Fields on page 169.

SEE ALSO:

Create a Batch with Binary Attachments

Get Information for a Batch

Monitor a Batch

Get Information for All Batches in a Job

Interpret Batch State

Get a Batch Request

Get Batch Results

Work with Jobs

Job and Batch Lifespan

Limits

About URIs

BatchInfo

Quick Start: Bulk API 2.0

Monitor a Batch

You can monitor a Bulk API batch in Salesforce.

To track the status of bulk data load jobs and their associated batches, from Setup, in the `Quick Find` box, enter *`Bulk Data Load Jobs`*, then select **Bulk Data Load Jobs**. Click the **Job ID** to view the job detail page.

The job detail page includes a related list of all the batches for the job. The related list provides **View Request** and **View Response** links for each batch. If the batch is a CSV file, the links return the request or response in CSV format. If the batch is an XML or JSON file, the links return the request or response in XML or JSON format, respectively. These links are available for batches created in API version 19.0 and later. For Bulk V2 type jobs, batch information is unavailable.

For more information, see "Monitor Bulk Data Load Jobs" in the Salesforce online help.

SEE ALSO:

- Get Information for a Batch
- Add a Batch to a Job
- Get Information for All Batches in a Job
- Interpret Batch State
- Get a Batch Request
- Get Batch Results
- Handle Failed Records in Batches
- Work with Jobs
- Job and Batch Lifespan
- Limits
- About URIs
- BatchInfo
- Quick Start: Bulk API 2.0

Get Information for a Batch

Get information about an existing batch by sending a GET request to this URI.

URI

/services/async/***APIversion***/job/***jobid***/batch/***batchId***

Example request body

No request body is allowed.

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <id>751D0000000004rIAA</id>
 <jobId>750D0000000002lIAA</jobId>
 <state>InProgress</state>
 <createdDate>2009-04-14T18:15:59.000Z</createdDate>
 <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
 <numberRecordsProcessed>0</numberRecordsProcessed>
 <numberRecordsFailed>0</numberRecordsFailed>
 <totalProcessingTime>0</totalProcessingTime>
 <apiActiveProcessingTime>0</apiActiveProcessingTime>
 <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Example JSON response body

```
{
   "apexProcessingTime" : 0,
   "apiActiveProcessingTime" : 0,
   "createdDate" : "2015-12-15T22:52:49.000+0000",
   "id" : "751D00000004YGeIAM",
   "jobId" : "750D00000004SkVIAU",
   "numberRecordsFailed" : 0,
   "numberRecordsProcessed" : 0,
   "state" : "InProgress",
   "systemModstamp" : "2015-12-15T22:52:49.000+0000",
   "totalProcessingTime" : 0
}
```

SEE ALSO:

Add a Batch to a Job

Monitor a Batch

Get Information for All Batches in a Job

Interpret Batch State

Get a Batch Request

Get Batch Results

Job and Batch Lifespan

Limits

BatchInfo

About URIs

Work with Jobs

Quick Start: Bulk API 2.0

Get Information for All Batches in a Job

Get information about all batches in a job by sending a GET request to this URI.

URI

`/services/async/`***`APIversion`***`/job/`***`jobid`***`/batch`

Method

GET

Example request body

No request body is allowed.

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfoList
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <batchInfo>
  <id>751D0000000004rIAA</id>
  <jobId>750D0000000002lIAA</jobId>
  <state>InProgress</state>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:16:09.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D0000000004sIAA</id>
  <jobId>750D0000000002lIAA</jobId>
  <state>InProgress</state>
  <createdDate>2009-04-14T18:16:00.000Z</createdDate>
  <systemModstamp>2009-04-14T18:16:09.000Z</systemModstamp>
  <numberRecordsProcessed>800</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>5870</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>2166</apexProcessingTime>
 </batchInfo>
</batchInfoList>
```

Example JSON response body

```
{
   "batchInfo" : [
      {
         "apexProcessingTime" : 0,
         "apiActiveProcessingTime" : 0,
         "createdDate" : "2015-12-15T21:56:43.000+0000",
         "id" : "751D00000004YGZIA2",
         "jobId" : "750D00000004SkVIAU",
         "numberRecordsFailed" : 0,
         "numberRecordsProcessed" : 0,
         "state" : "Queued",
         "systemModstamp" : "2015-12-15T21:57:19.000+0000",
         "totalProcessingTime" : 0
      },
      {
         "apexProcessingTime" : 0,
         "apiActiveProcessingTime" : 2166,
         "createdDate" : "2015-12-15T22:52:49.000+0000",
         "id" : "751D00000004YGeIAM",
         "jobId" : "750D00000004SkVIAU",
         "numberRecordsFailed" : 0,
         "numberRecordsProcessed" : 800,
         "state" : "Completed",
         "systemModstamp" : "2015-12-15T22:54:54.000+0000",
         "totalProcessingTime" : 5870
      }
```

```
    ]
}
```

SEE ALSO:

- Add a Batch to a Job
- Monitor a Batch
- Get Information for a Batch
- Interpret Batch State
- Get a Batch Request
- Get Batch Results
- Job and Batch Lifespan
- Limits
- BatchInfo
- About URIs
- Work with Jobs
- Quick Start: Bulk API 2.0

Interpret Batch State

This list gives you more details about the various states, also known as statuses, of a batch. The batch state informs you whether to proceed to get the results, or whether you must wait or fix errors related to your request.

`Queued`

Processing of the batch hasn't started yet. If the job associated with this batch is aborted, the batch isn't processed and its state is set to `NotProcessed`.

`InProgress`

The batch is being processed. If the job associated with the batch is aborted, the batch is still processed to completion. You must close the job associated with the batch so that the batch can finish processing.

`Completed`

The batch has been processed completely, and the result resource is available. The result resource indicates if some records failed. A batch can be completed even if some or all the records failed. If a subset of records failed, the successful records aren't rolled back.

`Failed`

The batch failed to process the full request due to an unexpected error, such as the request is compressed with an unsupported format, or an internal server error. Even if the batch failed, some records could have completed successfully. If the `numberRecordsProcessed` field in the response is greater than zero, you should get the results to see which records were processed, and if they were successful.

`NotProcessed`

The batch won't be processed. This state is assigned when a job is aborted while the batch is queued. For bulk queries, if the job has PK chunking enabled, this state is assigned to the original batch that contains the query when the subsequent batches are created.

After the original batch is changed to this state, you can monitor the subsequent batches and retrieve each batch's results when it's completed.

SEE ALSO:

Add a Batch to a Job

Monitor a Batch

Get Information for All Batches in a Job

Get a Batch Request

Get Batch Results

Handle Failed Records in Batches

Job and Batch Lifespan

Limits

BatchInfo

About URIs

Work with Jobs

Quick Start: Bulk API 2.0

Get a Batch Request

Get a batch request by sending a GET request to the following URI. This is available in API version 19.0 and later.

Alternatively, you can get a batch request in Salesforce. To track the status of bulk data load jobs and their associated batches, from Setup, in the `Quick Find` box, enter *`Bulk Data Load Jobs`*, then select **Bulk Data Load Jobs**. Click the **Job ID** to view the job detail page. The job detail page includes a related list of all the batches for the job. The related list provides **View Request** and **View Response** links for each batch. If the batch is a CSV file, the links return the request or response in CSV format. If the batch is an XML or JSON file, the links return the request or response in XML or JSON format, respectively.

URI

`/services/async/`***`APIversion`***`/job/`***`jobid`***`/batch/`***`batchId`***`/request`

Example request body

No request body is allowed.

Example XML response body

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 0 (batch 0)</name>
  </sObject>
  <sObject>
    <description>Created from Bulk API</description>
    <name>[Bulk API] Account 1 (batch 0)</name>
  </sObject>
</sObjects>
```

Example JSON response body

```
[
   {
        "Name" : "[Bulk API] Account 0 (batch 0)",
        "description" : "Created from Bulk API"
    },
    {
        "Name" : "[Bulk API] Account 1 (batch 0)",
        "description" : "Created from Bulk API"
    }
]
```

SEE ALSO:

- Get Information for a Batch
- Monitor a Batch
- Get Information for All Batches in a Job
- Interpret Batch State
- Get Batch Results
- Work with Jobs
- Job and Batch Lifespan
- Limits
- About URIs
- BatchInfo
- Quick Start: Bulk API 2.0

Get Batch Results

Get results of a batch that completed processing by sending a GET request to this URI. If the batch is a CSV file, the response is in CSV format. If the batch is an XML or JSON file, the response is in XML or JSON format, respectively.

Alternatively, you can monitor a Bulk API batch in Salesforce. To track the status of bulk data load jobs and their associated batches, from Setup, in the `Quick Find` box, enter *`Bulk Data Load Jobs`*, then select **Bulk Data Load Jobs**. Click the **Job ID** to view the job detail page.

The job detail page includes a related list of all the batches for the job. The related list provides **View Request** and **View Response** links for each batch. If the batch is a CSV file, the links return the request or response in CSV format. If the batch is an XML or JSON file, the links return the request or response in XML or JSON format, respectively. These links are available for batches created in API version 19.0 and later. For Bulk V2 type jobs, batch information is unavailable. The **View Response** link returns the same results as the following URI resource.

URI

`/services/async/`***`APIversion`***`/job/`***`jobid`***`/batch/`***`batchId`***`/result`

Example request body

No request body is allowed.

Example response body

For an XML batch

```
<?xml version="1.0" encoding="UTF-8"?>
<results xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <result>
    <id>001D000000ISUr3IAH</id>
    <success>true</success>
    <created>true</created>
  </result>
  <result>
    <id>001D000000ISUr4IAH</id>
    <success>true</success>
    <created>true</created>
  </result>
</results>
```

For a JSON batch

```
[
   {
      "success" : true,
      "created" : true,
      "id" : "001xx000003DHP0AAO",
      "errors" : []
   },
   {
      "success" : true,
      "created" : true,
      "id" : "001xx000003DHP1AAO",
      "errors" : []
   }
]
```

For a CSV batch

```
"Id","Success","Created","Error"
"003D000000Q89kQIAR","true","true",""
"003D000000Q89kRIAR","true","true",""
"","false","false","REQUIRED_FIELD_MISSING:Required fields are missing:
[LastName]:LastName --"
```

Note: The batch result indicates that the last record was not processed successfully because the `LastName` field was missing. The `Error` column includes error information. You must look at the `Success` field for each result row to ensure that all rows were processed successfully. For more information, see Handle Failed Records in Batches on page 123.

SEE ALSO:

- Add a Batch to a Job
- Monitor a Batch
- Get a Batch Request
- Get Information for a Batch
- Get Information for All Batches in a Job
- Interpret Batch State
- Job and Batch Lifespan
- Limits
- BatchInfo
- About URIs
- Work with Jobs
- Quick Start: Bulk API 2.0

Handle Failed Records in Batches

A batch can have a `Completed` state even if some or all of the records failed. If a subset of records failed, the successful records aren't rolled back. Likewise, even if the batch has a `Failed` state or if a job is aborted, some records could have completed successfully.

When you get the batch results, it's important to look at the `Success` field for each result row to ensure that all rows were processed successfully. If a record wasn't processed successfully, the `Error` column includes more information about the failure.

To identify failed records and log them to an error file:

1. Wait for the batch to finish processing. See Get Information for a Batch on page 116 and Interpret Batch State on page 119.
2. Get the batch results.

   This sample CSV batch result shows an error for the last record because the `LastName` field was missing:

   ```
   "Id","Success","Created","Error"
   "003D000000Q89kQIAR","true","true",""
   "003D000000Q89kRIAR","true","true",""
   "","false","false","REQUIRED_FIELD_MISSING:Required fields are missing:
   [LastName]:LastName --"
   ```

3. Parse the results for each record:
   a. Track the record number for each result record. Each result record corresponds to a record in the batch. The results are returned in the same order as the records in the batch request. It's important to track the record number in the results so that you can identify the associated failed record in the batch request.
   b. If the `Success` field is `false`, the row wasn't processed successfully. Otherwise, the record was processed successfully, and you can proceed to check the result for the next record.
   c. Get the contents of the `Error` column.

d. Write the contents of the corresponding record in the batch request to an error file on your computer. Append the information from the `Error` column. If you don't cache the batch request that you submitted, you can retrieve the batch request from Salesforce.

After you examine each result record, you can manually fix each record in the error file and submit these records in a new batch. To check that each record processed successfully, repeat the previous steps.

SEE ALSO:

Add a Batch to a Job

Errors

Limits

Bulk API Reference

These are the supported resources for the Bulk API, as well as the details on errors and processing limits.

Schema

The Bulk API service is described by an XML Schema Document (XSD) file.

JobInfo

A job contains one or more batches of data for you to submit to Salesforce for processing. When a job is created, Salesforce sets the job state to `Open`.

BatchInfo

A BatchInfo contains one batch of data for you to submit to Salesforce for processing.

Headers

These are the custom HTTP request and response headers that are used for Bulk API.

Errors

Operations that you perform with Bulk API can trigger error codes. This list shows the most common error codes and the Bulk API action that possibly triggered them.

Schema

The Bulk API service is described by an XML Schema Document (XSD) file.

You can download the schema file for an API version by using this URI:

`Web_Services_SOAP_endpoint_hostname``/services/async/`*`APIversion`*`/AsyncApi.xsd`

For example, if you're working with version 55.0 of the Bulk API, the URI for a production org is in this format.

`https://`***`MyDomainName`***`.my.salesforce.com/services/async/55.0/AsyncApi.xsd`

You can find the My Domain name and My Domain login URL for your org on the My Domain page in Setup. Or, to get the hostname of your My Domain login URL in Apex, use the `getMyDomainHostname()` method of the `System.DomainCreator` class.

Schema and API Versions

The schema file is available for API versions earlier than the current release. You can download the schema file for API version 18.0 and later. For example, if you want to download the schema file for API version 31.0, use this URI:

```
https://MyDomainName.my.salesforce.com/services/async/31.0/AsyncApi.xsd
```

SEE ALSO:

- JobInfo
- BatchInfo
- Errors

JobInfo

A job contains one or more batches of data for you to submit to Salesforce for processing. When a job is created, Salesforce sets the job state to `Open`.

You can create a new job, get information about a job, close a job, or abort a job using the JobInfo resource.

Fields

Name	Type	Request	Description
`apiVersion`	string	Read only. Do not set for new job.	The API version of the job set in the URI when the job was created. The earliest supported version is 17.0.
`apexProcessingTime`	long	Do not specify for new job.	The number of milliseconds taken to process triggers and other processes related to the job data. This is the sum of the equivalent times in all batches in the job. This doesn't include the time used for processing asynchronous and batch Apex operations. If there are no triggers, the value is `0`. See also `apiActiveProcessingTime` and `totalProcessingTime`. This field is available in API version 19.0 and later.
`apiActiveProcessingTime`	long	Do not specify for new job.	The number of milliseconds taken to actively process the job. It 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. This is the sum of the equivalent times in all batches in the job. See also `apexProcessingTime` and `totalProcessingTime`. This field is available in API version 19.0 and later.
`assignmentRuleId`	string	Can't update after creation.	The ID of a specific assignment rule to run for a case or a lead. The assignment rule can be active or inactive. The ID can be retrieved by using the SOAP-based SOAP API to query the AssignmentRule object.

Name	Type	Request	Description
`concurrencyMode`	ConcurrencyModeEnum		The concurrency mode for the job. The valid values are: • `Parallel`: Process batches in parallel mode. This is the default value. • `Serial`: Process batches 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 batches are processed one at a time, but can significantly increase the processing time.
`contentType`	ContentType		The content type for the job. The valid values are: • `CSV`—data in CSV format (default and only supported content type for Bulk V2 type jobs) • `JSON`—data in JSON format • `XML`—data in XML format (default option for Bulk V1 type jobs) • `ZIP_CSV`—data in CSV format in a zip file containing binary attachments • `ZIP_JSON`—data in JSON format in a zip file containing binary attachments • `ZIP_XML`—data in XML format in a zip file containing binary attachments
`createdById`	string	System field	The ID of the user who created this job. All batches must be created by this same user.
`createdDate`	dateTime	System field	The date and time in the UTC time zone when the job was created.
`externalIdFieldName`	string	Required with upsert	The name of the external ID field for an `upsert()`.
`id`	string	Do not specify for new job.	Unique ID for this job. All GET operations return this value in results.
`numberBatchesCompleted`	int	Do not specify for new job.	The number of batches that have been completed for this job.
`numberBatchesQueued`	int	Do not specify for new job.	The number of batches queued for this job.
`numberBatchesFailed`	int	Do not specify for new job.	The number of batches that have failed for this job.
`numberBatchesInProgress`	int	Do not specify for new job.	The number of batches that are in progress for this job.

Name	Type	Request	Description
`numberBatchesTotal`	int	Do not specify for new job.	The number of total batches currently in the job. This value increases as more batches are added to the job. When the job `state` is `Closed` or `Failed`, this number represents the final total. The job is complete when `numberBatchesTotal` equals the sum of `numberBatchesCompleted` and `numberBatchesFailed`.
`numberRecordsFailed`	int	Do not specify for new job.	The number of records that were not processed successfully in this job. This field is available in API version 19.0 and later.
`numberRecordsProcessed`	int	Do not specify for new job.	The number of records already processed. This number increases as more batches are processed.
`numberRetries`	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.
`object`	string	Required	The object type for the data being processed. All data in a job must be of a single object type.
`operation`	`OperationEnum`	Required	The processing operation for all the batches in the job. The valid values are: • `delete` • `hardDelete` • `insert` • `query` • `queryAll` • `update` • `upsert` Warning: The operation value must match that shown here. For example, you get an error if you use `INSERT` instead of `insert`. To ensure referential integrity, the delete operation supports cascading deletions. If you delete a parent record, you delete its children automatically, as long as each child record can be deleted. For example, if you delete a Case record, the Bulk API automatically deletes any child records, such as CaseComment, CaseHistory, and CaseSolution records associated with that case. However, if a CaseComment is not deletable or is currently being used, then the delete operation on the parent Case record fails.

Name	Type	Request	Description
			Warning: When the `hardDelete` value is specified, the deleted records aren't stored in the Recycle Bin. Instead, they become immediately eligible for deletion. The permission for this operation, "Bulk API Hard Delete," is disabled by default and must be enabled by an administrator. A Salesforce user license is required for hard delete.
`state`	`JobStateEnum`	Required if creating, closing, or aborting a job.	The current state of processing for the job: • `Open`: The job has been created, and data can be added to the job. • `Closed`: No new data can be added to this job. Data associated with the job may be processed after a job is closed. You cannot 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. • `Failed`: The job has failed. Batches that were successfully processed can't be rolled back. The BatchInfoList contains a list of all batches for the job. From the results of BatchInfoList, results can be retrieved for completed batches. The results indicate which records have been processed. The `numberRecordsFailed` field contains the number of records that were not processed successfully.
`systemModstamp`	dateTime	System field	Date and time in the UTC time zone when the job finished.
`totalProcessingTime`	long	Do not specify for new job.	The number of milliseconds taken to process the job. This is the sum of the total processing times for all batches in the job. See also `apexProcessingTime` and `apiActiveProcessingTime`. This field is available in API version 19.0 and later.

SEE ALSO:

Work with Jobs

Quick Start: Bulk API 2.0

SOAP API Developer Guide

BatchInfo

A BatchInfo contains one batch of data for you to submit to Salesforce for processing.

BatchInfo

Name	Type	Request	Description
`apexProcessingTime`	long	System field	The number of milliseconds taken to process triggers and other processes related to the batch data. If there are no triggers, the value is `0`. This doesn't include the time used for processing asynchronous and batch Apex operations. See also `apiActiveProcessingTime` and `totalProcessingTime`. This field is available in API version 19.0 and later.
`apiActiveProcessingTime`	long	System field	The number of milliseconds taken to actively process the batch, and includes `apexProcessingTime`. This doesn't include the time the batch waited in the queue to be processed or the time required for serialization and deserialization. See also `totalProcessingTime`. This field is available in API version 19.0 and later.
`createdDate`	dateTime	System field	The date and time in the UTC time zone when the batch was created. This is not the time processing began, but the time the batch was added to the job.
`id`	string	Required	The ID of the batch. May be globally unique, but does not have to be.
`jobId`	string	Required	The unique, 18–character ID for the job associated with this batch.
`numberRecordsFailed`	int	System field	The number of records that were not processed successfully in this batch. This field is available in API version 19.0 and later.
`numberRecordsProcessed`	int	System field	The number of records processed in this batch at the time the request was sent. This number increases as more batches are processed.
`state`	`BatchStateEnum`	System field	The current state of processing for the batch: • `Queued`: Processing of the batch hasn't started yet. If the job associated with this batch is aborted, the batch isn't processed and its state is set to `NotProcessed`. • `InProgress`: The batch is being processed. If the job associated with the batch is aborted, the batch is still processed to completion. You must close the job associated with the batch so that the batch can finish processing. • `Completed`: The batch has been processed completely, and the result resource is available. The result resource indicates if

Name	Type	Request	Description
			some records failed. A batch can be completed even if some or all the records failed. If a subset of records failed, the successful records aren't rolled back. • `Failed`: The batch failed to process the full request due to an unexpected error, such as the request is compressed with an unsupported format, or an internal server error. The `stateMessage` element could contain more details about any failures. Even if the batch failed, some records could have completed successfully. The `numberRecordsProcessed` field tells you how many records were processed. The `numberRecordsFailed` field contains the number of records that were not processed successfully. • `NotProcessed`: The batch won't be processed. This state is assigned when a job is aborted while the batch is queued. For bulk queries, if the job has PK chunking enabled, this state is assigned to the original batch that contains the query when the subsequent batches are created. After the original batch is changed to this state, you can monitor the subsequent batches and retrieve each batch's results when it's completed. Then you can safely close the job.
`stateMessage`	string	System field	When the `state` value is `Failed`, this field contains the reasons for failure. If there are multiple failures, the message may be truncated. If so, fix the known errors and re-submit the batch. Even if the batch failed, some records could have completed successfully.
`systemModstamp`	dateTime	System field	The date and time in the UTC time zone that processing ended. This is only valid when the state is Completed.
`totalProcessingTime`	long	System field	The number of milliseconds taken to process the batch. This excludes the time the batch waited in the queue to be processed. See also `apexProcessingTime` and `apiActiveProcessingTime`. This field is available in API version 19.0 and later.

HTTP BatchInfoList

Name	Type	Description
`batchInfo`	BatchInfo	One BatchInfo resource for each batch in the associated job. For the structure of BatchInfo, see BatchInfo on page 129.

SEE ALSO:

Work with Batches

Interpret Batch State

Quick Start: Bulk API 2.0

SOAP API Developer Guide

Headers

These are the custom HTTP request and response headers that are used for Bulk API.

- Content Type Header
- Batch Retry Header
- Line Ending Header
- PK Chunking Header

Content Type Header
Use the Content Type header to specify the format for your request and response. Set the value of this header to match the `contentType` of the job you're working with.

Batch Retry Header

Line Ending Header

PK Chunking Header
Use the primary key (PK) chunking request header to enable automatic PK chunking for a bulk query job. PK chunking splits bulk queries on large tables into chunks based on the record IDs, or primary keys, of the queried records.

Warning Header
This header is returned if there are warnings, such as the use of a deprecated version of the API.

Sforce Call Options Header
Use the Sforce Call Options header to specify client-specific options when accessing Bulk API resources.

Content Type Header

Use the Content Type header to specify the format for your request and response. Set the value of this header to match the `contentType` of the job you're working with.

For jobs with a `contentType` of CSV, XML is used as the response format except in the case of bulk query results, which are returned in CSV. To ensure that you retrieve responses in JSON, create a job with a `contentType` of JSON and use JSON for your batch payloads. To ensure that you retrieve responses in XML, create a job with a `contentType` of XML or CSV, and use the same format for your batch payloads.

If the job's `contentType` is unavailable, for example, when you create a job or when you submit a request with a bad job ID, the response respects the value of this header. If this header isn't included, the response defaults to XML.

Header Field Name and Values

Field name

`Content-Type`

Field values

- `application/json` (JSON is the preferred format.)
- `application/xml` (XML is the preferred format.)
- `text/csv` (CSV is the preferred format. Except for bulk query results, responses are returned in XML.)

Example

`Content-Type: application/json`

Batch Retry Header

When you create a bulk job, the Batch Retry request header lets you disable retries for unfinished batches included in the job. Use this header to limit the batch processing time for batches that consistently time out.

Header Field Name and Values

Field name

`Sforce-Disable-Batch-Retry`

Field values

- `TRUE`. Unfinished batches in this job aren't retried.
- `FALSE`. Unfinished batches in this job are retried the standard number of times (15 for bulk queries and 10 for bulk uploads). If the header isn't provided in the request, this is the default value.

Example

`Sforce-Disable-Batch-Retry: TRUE`

Line Ending Header

When you're creating a bulk upload job, the Line Ending request header lets you specify whether line endings are read as line feeds (LFs) or as carriage returns and line feeds (CRLFs) for fields of type `Text Area` and `Text Area (Long)`.

Header Field Name and Values

Field name

`Sforce-Line-Ending`

Field values

- `LF`. Line endings are read as LFs.
- `CRLF`. Line endings are read as CRLFs.

Example

`Sforce-Line-Ending: CRLF`

PK Chunking Header

Use the primary key (PK) chunking request header to enable automatic PK chunking for a bulk query job. PK chunking splits bulk queries on large tables into chunks based on the record IDs, or primary keys, of the queried records.

Each chunk is processed as a separate batch that counts toward your daily batch limit, and you must download each batch's results separately. PK chunking works only with queries that don't include subqueries or conditions other than `WHERE`.

PK chunking works by adding record ID boundaries to the query with a `WHERE` clause, limiting the query results to a smaller chunk of the total results. The remaining results are fetched with extra queries that contain successive boundaries. The number of records within the ID boundaries of each chunk is referred to as the chunk size. The first query retrieves records between a specified starting ID and the starting ID plus the chunk size. The next query retrieves the next chunk of records, and so on.

For example, let's say you enable PK chunking for the following query on an Account table with 10,000,000 records.

```
SELECT Name FROM Account
```

Assuming a chunk size of 250,000 and a starting record ID of `001300000000000`, the query is split into these 40 queries. Each query is submitted as a separate batch.

```
SELECT Name FROM Account WHERE Id >= 001300000000000 AND Id < 00130000000132G
SELECT Name FROM Account WHERE Id >= 00130000000132G AND Id < 00130000000264W
SELECT Name FROM Account WHERE Id >= 00130000000264W AND Id < 00130000000396m
...
SELECT Name FROM Account WHERE Id >= 00130000000euQ4 AND Id < 00130000000fxSK
```

Each query executes on a chunk of 250,000 records specified by the base-62 ID boundaries.

PK chunking is designed for extracting data from entire tables, but you can also use it for filtered queries. Because records could be filtered from each query's results, the number of returned results for each chunk can be less than the chunk size. The IDs of soft-deleted records are counted when the query is split into chunks, but the records are omitted from the results. Therefore, if soft-deleted records fall within a given chunk's ID boundaries, the number of returned results is less than the chunk size. In some scenarios, the net chunk size can also be greater than what is specified.

The default chunk size is 100,000, and the maximum size is 250,000. The default starting ID is the first record in the table. However, you can specify a different starting ID to restart a job that failed between chunked batches.

When a query is successfully chunked, the original batch's status shows as `NOT_PROCESSED`. If the chunking fails, the original batch's status shows as `FAILED`, but any chunked batches that were successfully queued during the chunking attempt are processed as normal. When the original batch's status is changed to `NOT_PROCESSED`, monitor the subsequent batches. You can retrieve the results from each subsequent batch after it's completed. Then you can safely close the job.

Salesforce recommends that you enable PK chunking when querying tables with more than 10 million records or when a bulk query consistently times out. However, the effectiveness of PK chunking depends on the specifics of the query and the queried data.

Supported Objects

PK chunking only works with the following objects:

- Account
- AccountContactRelation
- AccountTeamMember
- AiVisitSummary
- Asset
- AssignedResource
- B2BMktActivity

- B2BMktProspect
- Campaign
- CampaignMember
- CandidateAnswer
- Case
- CaseArticle
- CaseComment
- ChangeRequest
- Claim
- ClaimParticipant
- Contact
- ContentDistribution
- ContentDocument
- ContentVersion
- ContractLineItem
- ConversationDefinitionEventLog
- ConversationEntry
- ConversationReason
- ConversationReasonExcerpt
- ConversationReasonGroup
- CustomerProperty
- EinsteinAnswerFeedback
- EmailMessage
- EngagementScore
- Event
- EventRelation
- FeedItem
- Incident
- Individual
- InsurancePolicy
- InsurancePolicyAsset
- InsurancePolicyParticipant
- Lead
- LeadInsight
- LinkedArticle
- LiveChatTranscript
- LoginHistory
- LoyaltyAggrPointExprLedger
- LoyaltyLedger
- LoyaltyMemberCurrency

- LoyaltyMemberTier
- LoyaltyPartnerProduct
- LoyaltyProgramMbrPromotion
- LoyaltyProgramMember
- LoyaltyProgramPartner
- LoyaltyProgramPartnerLedger
- MlRetrainingFeedback
- Note
- ObjectTerritory2Association
- Opportunity
- OpportunityContactRole
- OpportunityHistory
- OpportunityLineItem
- OpportunitySplit
- OpportunityTeamMember
- Order
- OrderItem
- Pricebook2
- PricebookEntry
- Problem
- Product2
- ProductConsumed
- ProductRequired
- QuickText Quote
- QuoteLineItem
- ReplyText
- ScoreIntelligence
- ServiceContract Task
- TaskRelation
- TermDocumentFrequency
- TransactionJournal
- User
- UserRole
- VoiceCall
- VoiceCallRecording
- Voucher
- WebCart
- WorkloadUnit
- WorkOrder
- WorkOrderLineItem

- WorkPlan
- WorkPlanTemplate

Support includes custom objects.

Sharing and History Objects

You can use PK chunking with any Sharing and History tables that support standard objects. To enable PK-Chunking for History and Sharing objects, specify the parent object in the `Sforce-Enable-PKChunking` header by using the `parent` header field name.

Filtered Queries

PK chunking is designed for extracting data from entire tables, but you can also use it for filtered queries.

Because records could be filtered from each query's results, the number of returned results for each chunk can be less than the chunk size. Also, the IDs of soft-deleted records are counted when the query is split into chunks, but the records are omitted from the results. Therefore, if soft-deleted records fall within a given chunk's ID boundaries, the number of returned results is less than the chunk size.

Some query limitations apply that *effectively disable PK chunking* for the specified bulk query job:

- Filtering on any field with "id" in the field name (ID fields).
- Using an "ORDER BY" clause.

Header Field Name and Values

Field name

`Sforce-Enable-PKChunking`

Field values

- `TRUE`—Enables PK chunking with the default chunk size, starting from the first record ID in the queried table.
- `FALSE`—Disables PK chunking. If the header isn't provided in the request, the default is `FALSE`.
- `chunkSize`—Specifies the number of records within the ID boundaries for each chunk. The default is 100,000, and the maximum size is 250,000. If the query contains filters or soft-deleted records, the number of returned results for each chunk could be less than the chunk size. Other factors could return a net chunk size that is greater than the specified `chunkSize`. To exercise a tighter limit on the number of records in a chunk, add the `useSampledData` header. In any case, consider experimenting to determine the optimal chunk size.
- `parent`—Specifies the parent object when you're enabling PK chunking for queries on sharing objects. The chunks are based on the parent object's records rather than the sharing object's records. For example, when querying on AccountShare, specify Account as the parent object. PK chunking is supported for sharing objects as long as the parent object is supported.

  Similarly, for CaseHistory, specify Case as the parent object. For example:

  ```
  Sforce-Enable-PKChunking: parent=Case
  ```
- `startRow`—Specifies the 15-character or 18-character record ID to be used as the lower boundary for the first chunk. Use this parameter to specify a starting ID when restarting a job that failed between batches.

Example

```
Sforce-Enable-PKChunking: chunkSize=50000; startRow=00130000000xEftMGH
```

Warning Header

This header is returned if there are warnings, such as the use of a deprecated version of the API.

Header Field Name and Values

Field name

`Warning`

Field values

- `warningCode`
- `warningMessage`

For warnings about deprecated API versions, the `warningCode` is 299.

Example

```
Warning: 299 - "This API is deprecated and will be removed by Summer '22. Please see
https://help.salesforce.com/articleView?id=000351312 for details."
```

Sforce Call Options Header

Use the Sforce Call Options header to specify client-specific options when accessing Bulk API resources.

Header Field Name and Values

Field name

`Sforce-Call-Options`

Field values

- `client`—A string that identifies a client, for use, for example, in event log files.

For an example of using `Sforce-Call-Options`, see Call Options Header in the *REST API Developer Guide*.

Errors

Operations that you perform with Bulk API can trigger error codes. This list shows the most common error codes and the Bulk API action that possibly triggered them.

ClientInputError

The operation failed with an unknown client-side error.

For binary attachments, the request content is provided both as an input stream and an attachment.

ExceededQuota

The job or batch you tried to create exceeds the allowed number for the past 24-hour period.

FeatureNotEnabled

Bulk API isn't enabled for this organization.

InvalidBatch

The batch ID specified in a batch update or query is invalid.

This error code is returned for binary attachments when the zip content is malformed or these conditions occur:

- The `request.txt` file can't be found, can't be read, is a directory, or contains invalid content.
- The decompressed size of a binary attachment is too large.
- The size of the zip file is too large.
- The total decompressed size of all the binary attachments is too large.

Note: A StatusCode of `INVALID_FIELD` is returned for the following conditions:

- A binary file referenced in the batch data is missing or is a directory.
- A binary file referenced in the batch data doesn't start with `#`.

For more information about binary attachment limits, see General Limits on page 7.

InvalidJob
The job ID specified in a query or update for a job, or a create, update, or query for batches is invalid.

The user attempted to create a job using a zip content type in API version 19.0 or earlier.

InvalidJobState
The job state specified in a job update operation is invalid.

InvalidOperation
The operation specified in a URI for a job is invalid. Check the spelling of "job" in the URI.

InvalidSessionId
The session ID specified is invalid.

InvalidUrl
The URI specified is invalid.

InvalidUser
Either the user sending an Bulk API request doesn't have the correct permission, or the job or batch specified was created by another user.

InvalidXML
XML contained in the request body is invalid.

Timeout
The connection timed out. This error is thrown if Salesforce takes too long to process a batch. For more information on timeout limits, see Batch processing time. If you get a timeout error when processing a batch, split your batch into smaller batches, and try again.

TooManyLockFailure
Too many lock failures while processing the current batch. This error may be returned during processing of a batch. To resolve, analyze the batches for lock conflicts. See General Limits on page 7.

Unknown
Exception with unknown cause occurred.

In addition, Bulk API uses the same status codes and exception codes as SOAP API. For more information on these codes, see "ExceptionCode" in the *SOAP API Developer Guide*.

SEE ALSO:

Handle Failed Records in Batches

Bulk API Query

Use bulk query to efficiently query large data sets and reduce the number of API requests. A bulk query can retrieve up to 15 GB of data, divided into 15 files of 1 GB each. The data formats supported are CSV, XML, and JSON.

How Bulk Queries Are Processed

The bulk query workflow begins when you create a bulk query job and then add one or more batches to the query job. When a bulk query is processed, Salesforce attempts to execute the query. If the query doesn't execute within the standard 2-minute timeout limit, the job fails and a QUERY_TIMEOUT error is returned. In this case, rewrite a simpler query, and resubmit the batch.

Use Bulk Query

When you add a batch to a bulk query job, the Content-Type in the header for the request must be `text/csv`, `application/xml`, or `application/json`, depending on the content type specified when the job was created. The actual SOQL statement supplied for the batch is in plain text format.

Walk Through a Bulk Query Sample

This code sample uses cURL to query several account records.

Walk Through a Bulk Query Sample Using PK Chunking

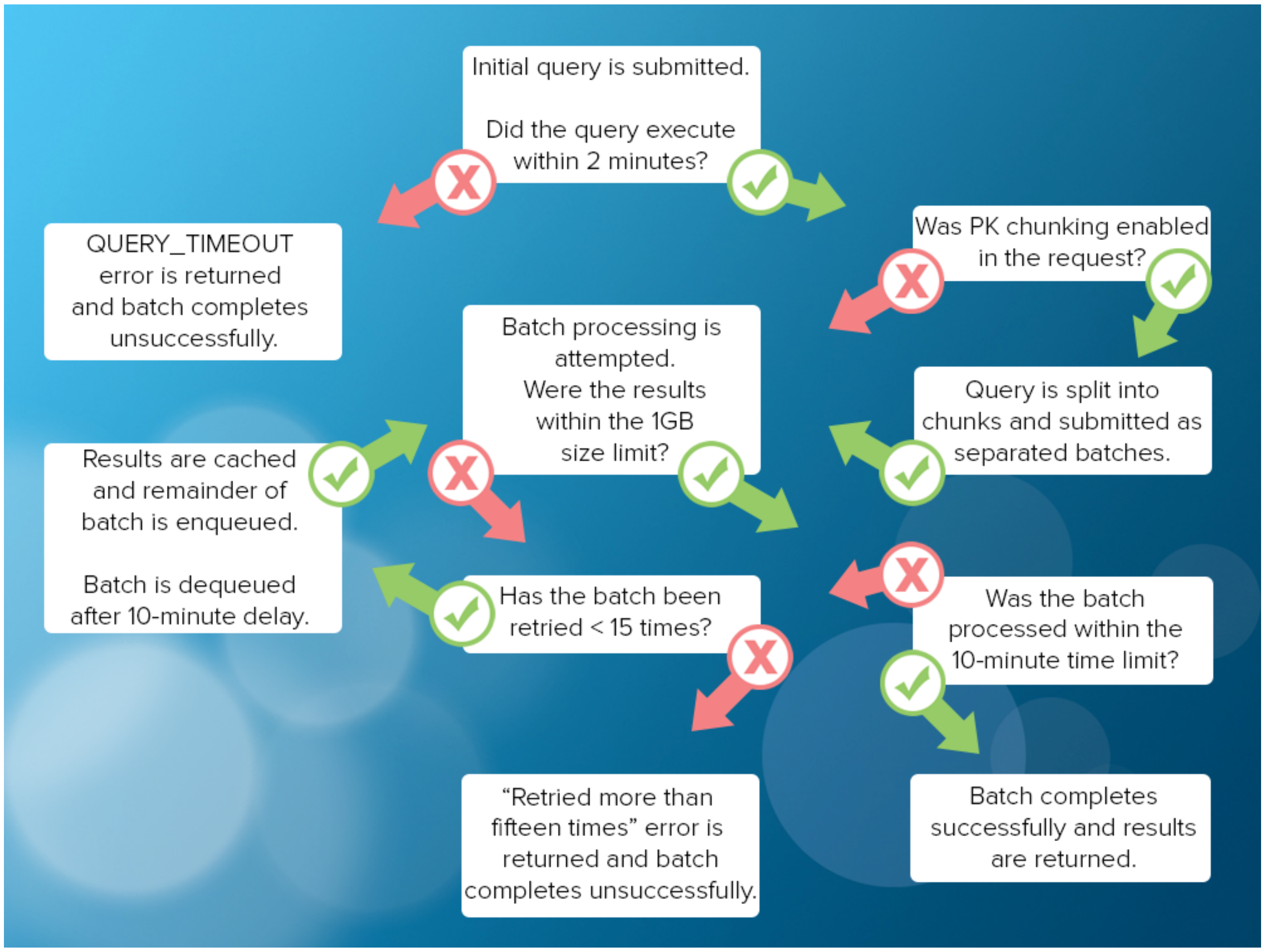
This code sample uses cURL to perform a bulk query with PK chunking enabled on several account records.

How Bulk Queries Are Processed

The bulk query workflow begins when you create a bulk query job and then add one or more batches to the query job. When a bulk query is processed, Salesforce attempts to execute the query. If the query doesn't execute within the standard 2-minute timeout limit, the job fails and a QUERY_TIMEOUT error is returned. In this case, rewrite a simpler query, and resubmit the batch.

If the query succeeds, Salesforce attempts to retrieve the results. If the results exceed the 1-GB file size limit or take longer than 10 minutes to retrieve, the completed results are cached and another attempt is made. After 15 attempts, the job fails and the error message "Retried more than fifteen times" is returned. In this case, consider using the PK Chunking header to split the query results into smaller chunks. If the attempts succeed, the results are returned and stored for 7 days.

This flowchart depicts how bulk queries are processed.


Initial query is submitted.
Did the query execute within 2 minutes?
QUERY_TIMEOUT error is returned and batch completes unsuccessfully.
Was PK chunking enabled in the request?
Batch processing is attempted. Were the results within the 1GB size limit?
Query is split into chunks and submitted as separated batches.
Results are cached and remainder of batch is enqueued.
Batch is dequeued after 10-minute delay.
Has the batch been retried < 15 times?
Was the batch processed within the 10-minute time limit?
"Retried more than fifteen times" error is returned and batch completes unsuccessfully.
Batch completes successfully and results are returned.

SEE ALSO:

- Use Bulk Query
- PK Chunking Header
- Walk Through a Bulk Query Sample

Use Bulk Query

When you add a batch to a bulk query job, the Content-Type in the header for the request must be `text/csv`, `application/xml`, or `application/json`, depending on the content type specified when the job was created. The actual SOQL statement supplied for the batch is in plain text format.

URI

/services/async/***APIversion***/job/***jobid***/batch

Bulk Query Request

```
POST baseURI/job/750x00000000014/batch
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
Content-Type: [either text/csv, application/xml, or application/json depending on job]
```

```
[plain text SOQL statement]
```

Bulk API query supports both query and queryAll operations. The queryAll operation returns records that have been deleted because of a merge or delete. The queryAll operation also returns information about archived Task and Event records. For more information, see queryAll() in the *SOAP API Developer Guide*.

Relationship queries traverse parent-to-child and child-to-parent relationships between objects to filter and return results. You can use SOQL relationships in bulk queries. For more information about SOQL relationships, see Using Relationship Queries in the *SOQL and SOSL Reference*.

Bulk API doesn't support queries with any of the following:

- GROUP BY, OFFSET, or TYPEOF clauses
- Aggregate functions such as `COUNT()`
- Date functions in GROUP BY clauses (date functions in WHERE clauses are supported)
- Compound address fields or compound geolocations fields

Example: **Requests, and Responses**

These are example Bulk Query requests and responses.

Create Bulk Query Batch HTTP Request

```
POST baseURI/job/750x00000000014/batch
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
Content-Type: text/csv; charset=UTF-8

SELECT Name, Description__c FROM Merchandise__c
```

Create Bulk Query Batch HTTP Response Body

```
<?xmlversion="1.0" encoding="UTF-8"?>
<batchInfo
  xmlns="http://www.force.com/2009/06/asyncapi/dataload">
    <id>751x00000000079AAA</id>
    <jobId>750x00000000014</jobId>
    <state>Queued</state>
    <createdDate>2009-09-01T17:44:45.000Z</createdDate>
    <systemModstamp>2009-09-01T17:44:45.000Z</systemModstamp>
    <numberRecordsProcessed>0</numberRecordsProcessed>
    <numberRecordsFailed>0</numberRecordsFailed>
    <totalProcessingTime>0</totalProcessingTime>
    <apiActiveProcessingTime>0</apiActiveProcessingTime>
    <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Get Batch Information for All Batches in a Job HTTP Request (used when PK chunking is enabled)

```
GET baseURI/job/750x00000000014/batch
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
```

Get Batch Information for All Batches in a Job HTTP Response Body

```
<?xml version="1.0" encoding="UTF-8"?><batchInfoList
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <batchInfo>
  <id>751D00000004YjwIAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>NotProcessed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk1IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk2IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk6IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk7IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>50000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
```

```
  <totalProcessingTime>500</totalProcessingTime>
  <apiActiveProcessingTime>500</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
</batchInfoList>
```

Get Batch Results HTTP Request

```
GET baseURI/job/750x00000000014/batch/751x00000000030/result
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
```

Get Batch Results HTTP Response Body

```
<result-list
xmlns="http://www.force.com/2009/06/asyncapi/dataload"><result>752x000000000F1</result></result-list>
```

Get Bulk Query Results HTTP Request

```
GET baseURI/job/750x00000000014/batch/751x00000000030/result/752x000000000F1
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...
```

Get Bulk Query Results HTTP Response Body

```
"Name", "Description__c"
"Merchandise1", "Description for merchandise 1"
"Merchandise2", "Description for merchandise 2"
```

XML Responses for Queries that Include ID

If you use XML for the ContentType of a Query Job, then a query that includes ID returns the ID field twice in the XML response data. Similarly, a query that does not include ID returns a single null ID field in the XML response data. For example, a query for `SELECT ID, FirstName, LastName FROM Contact` might return an XML response with records like:

```
<records xsi:type="sf:sObject" xmlns="urn:partner.soap.sforce.com">
    <sf:type>Contact</sf:type>
    <sf:Id>0038000000FrjoBQRW</sf:Id>
    <sf:Id>0038000000FrjoBQRW</sf:Id>
    <sf:FirstName>John</sf:FirstName>
    <sf:LastName>Smith</sf:LastName>
</records>
```

This is expected behavior and something to be aware of if you are accessing the full XML response data and not using WSC to access the web service response. For more information, see Queries and the Partner WSDL in the *SOAP API Developer Guide*.

Example: **Java Example Using WSC**

This example logs in to an organization using the Partner API, then instantiates a `BulkConnection` object using the service endpoint from the Partner API login.

```
public boolean login() {
  boolean success = false;

  String userId = getUserInput("UserID: ");
  String passwd = getUserInput("Password: ");
  String soapAuthEndPoint = "https://" + loginHost + soapService;
  String bulkAuthEndPoint = "https://" + loginHost + bulkService;
  try {
    ConnectorConfig config = new ConnectorConfig();
```

```
      config.setUsername(userId);
      config.setPassword(passwd);
      config.setAuthEndpoint(soapAuthEndPoint);
      config.setCompression(true);
      config.setTraceFile("traceLogs.txt");
      config.setTraceMessage(true);
      config.setPrettyPrintXml(true);
      System.out.println("AuthEndpoint: " +
          config.getRestEndpoint());
      PartnerConnection connection = new PartnerConnection(config);
      System.out.println("SessionID: " + config.getSessionId());
      config.setRestEndpoint(bulkAuthEndPoint);
      bulkConnection = new BulkConnection(config);
      success = true;
    } catch (AsyncApiException aae) {
      aae.printStackTrace();
    } catch (ConnectionException ce) {
      ce.printStackTrace();
    } catch (FileNotFoundException fnfe) {
      fnfe.printStackTrace();
    }
    return success;
  }

  public void doBulkQuery() {
    if ( ! login() ) {
      return;
    }
    try {
      JobInfo job = new JobInfo();
      job.setObject("Merchandise__c");

      job.setOperation(OperationEnum.query);
      job.setConcurrencyMode(ConcurrencyMode.Parallel);
      job.setContentType(ContentType.CSV);

      job = bulkConnection.createJob(job);
      assert job.getId() != null;

      job = bulkConnection.getJobStatus(job.getId());

      String query =
          "SELECT Name, Id, Description__c FROM Merchandise__c";

      long start = System.currentTimeMillis();

      BatchInfo info = null;
      ByteArrayInputStream bout =
          new ByteArrayInputStream(query.getBytes());
      info = bulkConnection.createBatchFromStream(job, bout);

      String[] queryResults = null;

      for(int i=0; i<10000; i++) {
        Thread.sleep(30000); //30 sec
        info = bulkConnection.getBatchInfo(job.getId(),
            info.getId());

        if (info.getState() == BatchStateEnum.Completed) {
          QueryResultList list =
              bulkConnection.getQueryResultList(job.getId(),
                  info.getId());
          queryResults = list.getResult();
          break;
        } else if (info.getState() == BatchStateEnum.Failed) {
          System.out.println("-------------- failed ----------"
              + info);
          break;
        } else {
          System.out.println("-------------- waiting ----------"
              + info);
        }
      }

      if (queryResults != null) {
        for (String resultId : queryResults) {
          bulkConnection.getQueryResultStream(job.getId(),
              info.getId(), resultId);
        }
      }
    } catch (AsyncApiException aae) {
      aae.printStackTrace();
    } catch (InterruptedException ie) {
      ie.printStackTrace();
    }
  }
```

SEE ALSO:

- How Bulk Queries Are Processed
- PK Chunking Header
- Walk Through a Bulk Query Sample

Walk Through a Bulk Query Sample

This code sample uses cURL to query several account records.

Note: Before you begin building an integration or other client application:

- Install your development platform according to its product documentation.
- Read through all the steps before creating the test client application. Also review the rest of this document to familiarize yourself with terms and concepts.

Create a Job

1. Create a file called `create-job.xml` containing this text.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <operation>query</operation>
  <object>Account</object>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
</jobInfo>
```

2. Using a command-line window, execute this cURL command to create a job.

   `curl -H "X-SFDC-Session: `***`sessionId`***`" -H "Content-Type: application/xml; charset=UTF-8" -d @create-job.xml https://`***`instance`***`.salesforce.com/services/async/55.0/job`

   `instance` is the portion of the `<serverUrl>` element and *`sessionId`* is the `<sessionId>` element that you noted in the login response.

   Salesforce returns an XML response with data such as this.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

Add a Batch to the Job

1. Create a file called `query.txt` to contain the SOQL query statement.

```
SELECT Id, Name FROM Account LIMIT 10
```

2. Using a command-line window, execute this cURL command to add a batch to the job:

   ```
   curl -d @query.txt -H "X-SFDC-Session: sessionId" -H "Content-Type: text/csv;
   charset=UTF-8" https://instance.salesforce.com/services/async/55.0/job/jobId/batch
   ```

   jobId is the job ID returned in the response to the job creation.

   Salesforce returns an XML response with data such as this.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <batchInfo
       xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <id>751x000000009vwAAA</id>
     <jobId>750x000000009tvAAA</jobId>
     <state>Queued</state>
     <createdDate>2016-01-10T00:59:47.000Z</createdDate>
     <systemModstamp>2016-01-10T00:59:47.000Z</systemModstamp>
     <numberRecordsProcessed>0</numberRecordsProcessed>
     <numberRecordsFailed>0</numberRecordsFailed>
     <totalProcessingTime>0</totalProcessingTime>
     <apiActiveProcessingTime>0</apiActiveProcessingTime>
     <apexProcessingTime>0</apexProcessingTime>
   </batchInfo>
   ```

   Note: When you add a batch to a bulk query job, the Content-Type in the header for the request must be `text/csv`, `application/xml`, or `application/json`, depending on the content type specified when the job was created. The actual SOQL statement supplied for the batch is in plain text format.

Check the Status of the Job and Batch

1. Using a command-line window, execute this cURL command to check the job status.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/55.0/job/jobId
   ```

   Salesforce returns an XML response with data such as this.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <jobInfo
       xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <id>750x000000009tvAAA</id>
     <operation>query</operation>
     <object>Account</object>
     <createdById>005x0000001WR0lAAG</createdById>
     <createdDate>2016-01-10T00:53:19.000Z</createdDate>
     <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
     <state>Open</state>
     <concurrencyMode>Parallel</concurrencyMode>
     <contentType>CSV</contentType>
     <numberBatchesQueued>0</numberBatchesQueued>
     <numberBatchesInProgress>0</numberBatchesInProgress>
     <numberBatchesCompleted>1</numberBatchesCompleted>
     <numberBatchesFailed>0</numberBatchesFailed>
     <numberBatchesTotal>1</numberBatchesTotal>
     <numberRecordsProcessed>10</numberRecordsProcessed>
     <numberRetries>0</numberRetries>
   ```

```
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

2. Using a command-line window, execute this cURL command to check the batch status.

   `curl -H "X-SFDC-Session: ` ***`sessionId`***`"`
   `https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***`/batch/`***`batchId`***

   `batchId` is the batch ID in the response to the batch creation.

   Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>10</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Retrieve the Results

1. Using the command-line window, execute this cURL command to retrieve the batch result list.

   `curl -H "X-SFDC-Session: ` ***`sessionId`***`"`
   `https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***`/batch/`***`batchId`***`/result`

   Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<result-list xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <result>752x00000004CJE</result>
</result-list>
```

   Note: If the batch required retries, there will be more than one `<result>` element in the output.

2. Using the command-line window, execute this cURL command to retrieve the results of the query.

   `curl -H "X-SFDC-Session: ` ***`sessionId`***`"`
   `https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***`/batch/`***`batchId`***`/result/`***`resultId`***

   `resultId` is the result ID in the response to the batch result list request.

Salesforce returns a CSV response with data such as this.

```
"Id","Name"
"001x000xxx4TU4JAAW","name161268--1296595660659"
"001x000xxx4TU4KAAW","name161269--1296595660659"
"001x000xxx4TU4LAAW","name161270--1296595660659"
"001x000xxx4TU4MAAW","name161271--1296595660659"
"001x000xxx4TU4NAAW","name161272--1296595660659"
"001x000xxx4TU4OAAW","name161273--1296595660659"
"001x000xxx4TU4PAAW","name161274--1296595660659"
"001x000xxx4TU4QAAW","name161275--1296595660659"
"001x000xxx4TU4RAAW","name161276--1296595660659"
"001x000xxx4TU4SAAW","name161277--1296595660659"
```

Close the Job

1. Create a file called `close-job.xml` containing this text.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <state>Closed</state>
</jobInfo>
```

2. Using a command-line window, execute this cURL command to close the job.

`curl -H "X-SFDC-Session: `***`sessionId`***`" -H "Content-Type: text/csv; charset=UTF-8" -d @close-job.xml https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***

Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Closed</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>1</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>1</numberBatchesTotal>
  <numberRecordsProcessed>10</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

SEE ALSO:

How Bulk Queries Are Processed

Use Bulk Query

Walk Through a Bulk Query Sample Using PK Chunking

Walk Through a Bulk Query Sample Using PK Chunking

This code sample uses cURL to perform a bulk query with PK chunking enabled on several account records.

Note: Before you begin building an integration or other client application:

- Install your development platform according to its product documentation.
- Read through all the steps before creating the test client application. Also review the rest of this document to familiarize yourself with terms and concepts.

Create a Job with PK Chunking Enabled

1. Create a file called `create-job.xml` containing this text.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <operation>query</operation>
  <object>Account</object>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
</jobInfo>
```

2. Using a command-line window, execute this cURL command to create a job with PK chunking enabled.

```
curl -H "X-SFDC-Session: sessionId" -H "Content-Type: application/xml; charset=UTF-8"
-H "Sforce-Enable-PKChunking: true" -d @create-job.xml
https://instance.salesforce.com/services/async/55.0/job
```

`instance` is the portion of the `<serverUrl>` element, and *`sessionId`* is the `<sessionId>` element that you noted in the login response.

Note: Salesforce recommends that you enable PK chunking when querying tables with more than 10 million records or when a bulk query consistently times out. For the purposes of this example, if you're querying significantly fewer records, set `chunkSize` to a number smaller than the number of records you're querying. For example, `Sforce-Enable-PKChunking: chunkSize=1000`. This way, you get to see PK chunking in action, and the query is split into multiple batches.

Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Open</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>0</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>0</numberBatchesTotal>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

Add a Batch to the Job

1. Create a file called `query.txt` to contain the SOQL query statement.

```
SELECT Id, Name FROM Account
```

2. Using a command-line window, execute this cURL command to add a batch to the job.

   `curl -d @query.txt -H "X-SFDC-Session: sessionId" -H "Content-Type: text/csv; charset=UTF-8" https://instance.salesforce.com/services/async/55.0/job/jobId/batch`

   `jobId` is the job ID returned in the response to the job creation.

   Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Queued</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T00:59:47.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Note: When you add a batch to a bulk query job, the Content-Type in the header for the request must be `text/csv`, `application/xml`, or `application/json`, depending on the content type specified when the job was created. The actual SOQL statement supplied for the batch is in plain text format.

Check the Status of the Job and Batch

1. Using a command-line window, execute this cURL command to check the job status.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/55.0/job/jobId
   ```

   Salesforce returns an XML response with data such as this.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <jobInfo
       xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <id>750x000000009tvAAA</id>
     <operation>query</operation>
     <object>Account</object>
     <createdById>005x0000001WR0lAAG</createdById>
     <createdDate>2016-01-10T00:53:19.000Z</createdDate>
     <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
     <state>Open</state>
     <concurrencyMode>Parallel</concurrencyMode>
     <contentType>CSV</contentType>
     <numberBatchesQueued>0</numberBatchesQueued>
     <numberBatchesInProgress>0</numberBatchesInProgress>
     <numberBatchesCompleted>4</numberBatchesCompleted>
     <numberBatchesFailed>0</numberBatchesFailed>
     <numberBatchesTotal>4</numberBatchesTotal>
     <numberRecordsProcessed>350000</numberRecordsProcessed>
     <numberRetries>0</numberRetries>
     <apiVersion>36.0</apiVersion>
     <numberRecordsFailed>0</numberRecordsFailed>
     <totalProcessingTime>3500</totalProcessingTime>
     <apiActiveProcessingTime>3500</apiActiveProcessingTime>
     <apexProcessingTime>0</apexProcessingTime>
   </jobInfo>
   ```

   Because PK chunking is enabled, extra batches are automatically created to process the entire query.

2. Using a command-line window, execute this cURL command to check the status of the original batch.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/55.0/job/jobId/batch/batchId
   ```

   `batchId` is the batch ID in the response to the batch creation.

   Salesforce returns an XML response with data such as this.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <batchInfo
       xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <id>751x000000009vwAAA</id>
     <jobId>750x000000009tvAAA</jobId>
     <state>Not Processed</state>
     <createdDate>2016-01-10T00:59:47.000Z</createdDate>
     <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
     <numberRecordsProcessed>0</numberRecordsProcessed>
     <numberRecordsFailed>0</numberRecordsFailed>
     <totalProcessingTime>0</totalProcessingTime>
     <apiActiveProcessingTime>0</apiActiveProcessingTime>
   ```

```
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>
```

Because PK chunking is enabled, the original batch is given a state of `Not Processed`. The query is processed in the remaining batches.

Get the IDs of the Remaining Batches

Using the command-line window, execute this cURL command to retrieve the remaining batches.

```
curl -H "X-SFDC-Session: sessionId"
https://instance.salesforce.com/services/async/55.0/job/jobId/batch
```

Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?><batchInfoList
   xmlns="http://www.force.com/2009/06/asyncapi/dataload">
 <batchInfo>
  <id>751D00000004YjwIAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>NotProcessed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk1IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk2IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
```

```
  <id>751D00000004Yk6IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>100000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>1000</totalProcessingTime>
  <apiActiveProcessingTime>1000</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
 <batchInfo>
  <id>751D00000004Yk7IAE</id>
  <jobId>750D00000004T5OIAU</jobId>
  <state>Completed</state>
  <createdDate>2016-01-10T00:59:47.000Z</createdDate>
  <systemModstamp>2016-01-10T01:00:19.000Z</systemModstamp>
  <numberRecordsProcessed>50000</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>500</totalProcessingTime>
  <apiActiveProcessingTime>500</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
 </batchInfo>
</batchInfoList>
```

Retrieve the Results

Perform these steps for each remaining batch.

1. Using the command-line window, execute this cURL command to retrieve the batch result list.

   `curl -H "X-SFDC-Session: `***`sessionId`***`"`
   `https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***`/batch/`***`batchId`***`/result`

   Salesforce returns an XML response with data such as this.

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <result-list xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <result>752x00000004CJE</result>
   </result-list>
   ```

   Note: If the batch required retries, there will be more than one `<result>` element in the output.

2. Using the command-line window, execute this cURL command to retrieve the results of the query.

   `curl -H "X-SFDC-Session: `***`sessionId`***`"`
   `https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***`/batch/`***`batchId`***`/result/`***`resultId`***

   `resultId` is the result ID in the response to the batch result list request.

   Salesforce returns a CSV response with data such as this.

   ```
   "Id","Name"
   "001x000xxx4TU4JAAW","name161268--1296595660659"
   "001x000xxx4TU4KAAW","name161269--1296595660659"
   "001x000xxx4TU4LAAW","name161270--1296595660659"
   ```

```
"001x000xxx4TU4MAAW","name161271--1296595660659"
"001x000xxx4TU4NAAW","name161272--1296595660659"
"001x000xxx4TU4OAAW","name161273--1296595660659"
"001x000xxx4TU4PAAW","name161274--1296595660659"
"001x000xxx4TU4QAAW","name161275--1296595660659"
"001x000xxx4TU4RAAW","name161276--1296595660659"
"001x000xxx4TU4SAAW","name161277--1296595660659"
...
```

Close the Job

1. Create a file called `close-job.xml` containing this text.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <state>Closed</state>
</jobInfo>
```

2. Using a command-line window, execute this cURL command to close the job.

`curl -H "X-SFDC-Session: `***`sessionId`***`" -H "Content-Type: text/csv; charset=UTF-8" -d @close-job.xml https://`***`instance`***`.salesforce.com/services/async/55.0/job/`***`jobId`***

Salesforce returns an XML response with data such as this.

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
    xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Account</object>
  <createdById>005x0000001WR0lAAG</createdById>
  <createdDate>2016-01-10T00:53:19.000Z</createdDate>
  <systemModstamp>2016-01-10T00:53:19.000Z</systemModstamp>
  <state>Closed</state>
  <concurrencyMode>Parallel</concurrencyMode>
  <contentType>CSV</contentType>
  <numberBatchesQueued>0</numberBatchesQueued>
  <numberBatchesInProgress>0</numberBatchesInProgress>
  <numberBatchesCompleted>4</numberBatchesCompleted>
  <numberBatchesFailed>0</numberBatchesFailed>
  <numberBatchesTotal>4</numberBatchesTotal>
  <numberRecordsProcessed>350000</numberRecordsProcessed>
  <numberRetries>0</numberRetries>
  <apiVersion>36.0</apiVersion>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>3500</totalProcessingTime>
  <apiActiveProcessingTime>3500</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</jobInfo>
```

SEE ALSO:

- How Bulk Queries Are Processed
- Use Bulk Query
- PK Chunking Header
- Walk Through a Bulk Query Sample

Sample Client Application Using Java

Use this code sample to create a test client application that inserts a number of account records using the REST-based Bulk API.

In addition to the step-by-step instructions that follow, the end of this section provides the complete code to make copying and pasting easier for you.

Note: Before you begin building an integration or other client application:

- Install your development platform according to its product documentation.
- Read through all the steps before creating the test client application. You may also wish to review the rest of this document to familiarize yourself with terms and concepts.

1. Set Up Your Client Application
   The Bulk API uses HTTP GET and HTTP POST methods to send and receive XML or JSON content, so it's simple to build clients in the language of your choice. This task uses a Java sample and the Salesforce Web Service Connector (WSC) toolkit provided by Salesforce to simplify development. WSC is a high-performing web service client stack implemented using a streaming parser. The toolkit has built-in support for the basic operations and objects used in the Bulk API.
2. Walk Through the Sample Code
   After you set up your client, you can build client applications that use the Bulk API. Use the sample to create a client application. Each section steps through part of the code. The complete sample is included at the end.

Set Up Your Client Application

The Bulk API uses HTTP GET and HTTP POST methods to send and receive XML or JSON content, so it's simple to build clients in the language of your choice. This task uses a Java sample and the Salesforce Web Service Connector (WSC) toolkit provided by Salesforce to simplify development. WSC is a high-performing web service client stack implemented using a streaming parser. The toolkit has built-in support for the basic operations and objects used in the Bulk API.

Review the library here:

`https://github.com/forcedotcom/wsc`

To download the Salesforce WSC toolkit:

1. Browse to `http://mvnrepository.com/artifact/com.force.api/force-wsc` or `https://repo1.maven.org/maven2/com/force/api/force-wsc/`
2. Navigate to the version of WSC that matches the API version you're using.
3. Click `force-wsc-`***`XX.X.X`***`-uber.jar`, and save the file to a local directory.

The Bulk API doesn't provide a login operation, so you must use the SOAP API to login.

To download the partner WSDL and compile it to Java classes with the WSC toolkit:

1. Log in to your Developer Edition Salesforce account. You must log in as an administrator or as a user who has the "Modify All Data" permission. Logins are checked to ensure they are from a known IP address. For more information, see Security and the API in the SOAP API Developer Guide.
2. From Setup, in the `Quick Find` box, enter "API", then select **API**.
3. Right-click **Partner WSDL** to display your browser's save options, and save the partner WSDL to a local directory. For information about the partner WSDL, see Using the Partner WSDL.
4. Compile the partner API code from the WSDL using the WSC compile tool:

   ```
   java -classpath pathToJar\force-wsc-XX.X.X-uber.jar com.sforce.ws.tools.wsdlc
   pathToWSDL\wsdlFilename .\wsdlGenFiles.jar
   ```

   For example, if force-wsc-*XX.X.X*-uber.jar is installed in `C:\salesforce\wsc`, and the partner WSDL is saved to `C:\salesforce\wsdl\partner`:

   ```
   java -classpath C:\salesforce\wsc\force-wsc-XX.X.X-uber.jar com.sforce.ws.tools.wsdlc
    C:\salesforce\wsdl\partner\partner.wsdl .\partner.jar
   ```

   force-wsc-*XX.X.X*-uber.jar and the generated `partner.jar` are the only libraries needed in the classpath for the code examples in the following sections.

Walk Through the Sample Code

After you set up your client, you can build client applications that use the Bulk API. Use the sample to create a client application. Each section steps through part of the code. The complete sample is included at the end.

This code sets up the packages and classes in the WSC toolkit and the code generated from the partner WSDL:

```
import java.io.*;
import java.util.*;

import com.sforce.async.*;
import com.sforce.soap.partner.PartnerConnection;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;
```

Set Up the `main()` Method

This code sets up the `main()` method for the class. It calls the `runSample()` method, which encompasses the processing logic for the sample. We look at the methods called in `runSample()` in subsequent sections.

```
    public static void main(String[] args)
      throws AsyncApiException, ConnectionException, IOException {
        BulkExample example = new BulkExample();
        // Replace arguments below with your credentials and test file name
        // The first parameter indicates that we are loading Account records
       example.runSample("Account", "myUser@myOrg.com", "myPassword", "mySampleData.csv");
```

```
    }

    /**
     * Creates a Bulk API job and uploads batches for a CSV file.
     */
    public void runSample(String sobjectType, String userName,
              String password, String sampleFileName)
            throws AsyncApiException, ConnectionException, IOException {
        BulkConnection connection = getBulkConnection(userName, password);
        JobInfo job = createJob(sobjectType, connection);
        List<BatchInfo> batchInfoList = createBatchesFromCSVFile(connection, job,
            sampleFileName);
        closeJob(connection, job.getId());
        awaitCompletion(connection, job, batchInfoList);
        checkResults(connection, job, batchInfoList);
    }
```

Login and Configure BulkConnection

This code logs in using a partner connection (PartnerConnection) and then reuses the session to create a Bulk API connection (BulkConnection).

```
    /**
     * Create the BulkConnection used to call Bulk API operations.
     */
    private BulkConnection getBulkConnection(String userName, String password)
          throws ConnectionException, AsyncApiException {
        ConnectorConfig partnerConfig = new ConnectorConfig();
        partnerConfig.setUsername(userName);
        partnerConfig.setPassword(password);
       partnerConfig.setAuthEndpoint("https://login.salesforce.com/services/Soap/u/55.0");

        // Creating the connection automatically handles login and stores
        // the session in partnerConfig
        new PartnerConnection(partnerConfig);
        // When PartnerConnection is instantiated, a login is implicitly
        // executed and, if successful,
        // a valid session is stored in the ConnectorConfig instance.
        // Use this key to initialize a BulkConnection:
        ConnectorConfig config = new ConnectorConfig();
        config.setSessionId(partnerConfig.getSessionId());
        // The endpoint for the Bulk API service is the same as for the normal
        // SOAP uri until the /Soap/ part. From here it's '/async/versionNumber'
        String soapEndpoint = partnerConfig.getServiceEndpoint();
        String apiVersion = "55.0";
        String restEndpoint = soapEndpoint.substring(0, soapEndpoint.indexOf("Soap/"))
            + "async/" + apiVersion;
        config.setRestEndpoint(restEndpoint);
        // This should only be false when doing debugging.
        config.setCompression(true);
        // Set this to true to see HTTP requests and responses on stdout
        config.setTraceMessage(false);
        BulkConnection connection = new BulkConnection(config);
        return connection;
    }
```

This BulkConnection instance is the base for using the Bulk API. The instance can be reused for the rest of the application lifespan.

Create a Job

After creating the connection, create a job. Data is always processed in the context of a job. The job specifies the details about the data being processed: which operation is being executed (insert, update, upsert, or delete) and the object type. This code creates a new insert job on the Account object.

```
    /**
     * Create a new job using the Bulk API.
     *
     * @param sobjectType
     *            The object type being loaded, such as "Account"
     * @param connection
     *            BulkConnection used to create the new job.
     * @return The JobInfo for the new job.
     * @throws AsyncApiException
     */
    private JobInfo createJob(String sobjectType, BulkConnection connection)
          throws AsyncApiException {
        JobInfo job = new JobInfo();
        job.setObject(sobjectType);
        job.setOperation(OperationEnum.insert);
        job.setContentType(ContentType.CSV);
        job = connection.createJob(job);
        System.out.println(job);
        return job;
    }
```

When a job is created, it’s in the `Open` state. In this state, new batches can be added to the job. When a job is `Closed`, batches can no longer be added.

Add Batches to the Job

Data is processed in a series of batch requests. Each request is an HTTP POST containing the data set in XML format in the body. Your client application determines how many batches are used to process the whole data set as long as the batch size and total number of batches per day are within the limits specified in Limits on page 78.

The processing of each batch comes with an overhead. Make batch sizes large enough to minimize the overhead processing cost, and small enough to be handled and transferred easily. Batch sizes between 1,000 and 10,000 records are considered reasonable.

This code splits a CSV file into smaller batch files and uploads them to Salesforce.

```
    /**
     * Create and upload batches using a CSV file.
     * The file into the appropriate size batch files.
     *
     * @param connection
     *            Connection to use for creating batches
     * @param jobInfo
     *            Job associated with new batches
     * @param csvFileName
     *            The source file for batch data
     */
    private List<BatchInfo> createBatchesFromCSVFile(BulkConnection connection,
          JobInfo jobInfo, String csvFileName)
            throws IOException, AsyncApiException {
        List<BatchInfo> batchInfos = new ArrayList<BatchInfo>();
        BufferedReader rdr = new BufferedReader(
            new InputStreamReader(new FileInputStream(csvFileName))
        );
        // read the CSV header row
        byte[] headerBytes = (rdr.readLine() + "\n").getBytes("UTF-8");
        int headerBytesLength = headerBytes.length;
        File tmpFile = File.createTempFile("bulkAPIInsert", ".csv");

        // Split the CSV file into multiple batches
        try {
            FileOutputStream tmpOut = new FileOutputStream(tmpFile);
            int maxBytesPerBatch = 10000000; // 10 million bytes per batch
            int maxRowsPerBatch = 10000; // 10 thousand rows per batch
            int currentBytes = 0;
            int currentLines = 0;
            String nextLine;
            while ((nextLine = rdr.readLine()) != null) {
                byte[] bytes = (nextLine + "\n").getBytes("UTF-8");
                // Create a new batch when our batch size limit is reached
                if (currentBytes + bytes.length > maxBytesPerBatch
                  || currentLines > maxRowsPerBatch) {
                    createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
                    currentBytes = 0;
                    currentLines = 0;
                }
                if (currentBytes == 0) {
                    tmpOut = new FileOutputStream(tmpFile);
                    tmpOut.write(headerBytes);
                    currentBytes = headerBytesLength;
                    currentLines = 1;
                }
                tmpOut.write(bytes);
                currentBytes += bytes.length;
                currentLines++;
            }
            // Finished processing all rows
            // Create a final batch for any remaining data
            if (currentLines > 1) {
                createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
            }
        } finally {
            tmpFile.delete();
        }
        return batchInfos;
    }
    /**
     * Create a batch by uploading the contents of the file.
     * This closes the output stream.
     *
     * @param tmpOut
     *            The output stream used to write the CSV data for a single batch.
     * @param tmpFile
     *            The file associated with the above stream.
     * @param batchInfos
     *            The batch info for the newly created batch is added to this list.
     * @param connection
     *            The BulkConnection used to create the new batch.
     * @param jobInfo
     *            The JobInfo associated with the new batch.
     */
    private void createBatch(FileOutputStream tmpOut, File tmpFile,
      List<BatchInfo> batchInfos, BulkConnection connection, JobInfo jobInfo)
              throws IOException, AsyncApiException {
        tmpOut.flush();
        tmpOut.close();
        FileInputStream tmpInputStream = new FileInputStream(tmpFile);
        try {
            BatchInfo batchInfo =
              connection.createBatchFromStream(jobInfo, tmpInputStream);
            System.out.println(batchInfo);
            batchInfos.add(batchInfo);

        } finally {
            tmpInputStream.close();
        }
    }
```

When the server receives a batch, it's immediately queued for processing. Errors in formatting aren't reported when sending the batch. These errors are reported in the result data when the batch is processed.

Tip: To import binary attachments, use the following methods. Specify the CSV, XML, or JSON content for the batch in the `batchContent` parameter, or include `request.txt` in the attached files and pass `null` to the `batchContent` parameter. These methods are contained within the `com.async.BulkConnection` class:

- `createBatchFromDir()`
- `createBatchWithFileAttachments()`
- `createBatchWithInputStreamAttachments()`
- `createBatchFromZipStream()`

Close the Job

After all batches are added to a job, close the job. Closing the job ensures that processing of all batches can finish.

```
    private void closeJob(BulkConnection connection, String jobId)
          throws AsyncApiException {
        JobInfo job = new JobInfo();
        job.setId(jobId);
        job.setState(JobStateEnum.Closed);
        connection.updateJob(job);
    }
```

Check Status on Batches

Batches are processed in the background. The size of the data set determines how long processing takes. During processing, you can retrieve and check the status of all batches, and you can see when processing is complete.

```
    /**
     * Wait for a job to complete by polling the Bulk API.
     *
     * @param connection
     *            BulkConnection used to check results.
     * @param job
     *            The job awaiting completion.
     * @param batchInfoList
     *            List of batches for this job.
     * @throws AsyncApiException
     */
    private void awaitCompletion(BulkConnection connection, JobInfo job,
          List<BatchInfo> batchInfoList)
            throws AsyncApiException {
        long sleepTime = 0L;
        Set<String> incomplete = new HashSet<String>();
        for (BatchInfo bi : batchInfoList) {
            incomplete.add(bi.getId());
        }
        while (!incomplete.isEmpty()) {
            try {
                Thread.sleep(sleepTime);
            } catch (InterruptedException e) {}
            System.out.println("Awaiting results..." + incomplete.size());
            sleepTime = 10000L;
            BatchInfo[] statusList =
              connection.getBatchInfoList(job.getId()).getBatchInfo();
            for (BatchInfo b : statusList) {
                if (b.getState() == BatchStateEnum.Completed
                  || b.getState() == BatchStateEnum.Failed) {
                    if (incomplete.remove(b.getId())) {
                        System.out.println("BATCH STATUS:\n" + b);
                    }
                }
            }
        }
    }
```

A batch is done when it's either failed or completed. This code loops infinitely until all the batches for the job have either failed or completed.

Get Results For a Job

You can retrieve the results of each batch when all batches are processed. Retrieve results whether the batch succeeded or failed, or even if the job was aborted, because only the result sets indicate the status of individual records. To properly pair a result with its corresponding record, the code must not lose track of how the batches correspond to the original data set. So keep the original list of batches from when they were created and use this list to retrieve results, as shown in this example:

```
    /**
     * Gets the results of the operation and checks for errors.
     */
    private void checkResults(BulkConnection connection, JobInfo job,
              List<BatchInfo> batchInfoList)
            throws AsyncApiException, IOException {
        // batchInfoList was populated when batches were created and submitted
        for (BatchInfo b : batchInfoList) {
            CSVReader rdr =
              new CSVReader(connection.getBatchResultStream(job.getId(), b.getId()));
            List<String> resultHeader = rdr.nextRecord();
            int resultCols = resultHeader.size();

            List<String> row;
            while ((row = rdr.nextRecord()) != null) {
                Map<String, String> resultInfo = new HashMap<String, String>();
                for (int i = 0; i < resultCols; i++) {
                    resultInfo.put(resultHeader.get(i), row.get(i));
                }
                boolean success = Boolean.valueOf(resultInfo.get("Success"));
                boolean created = Boolean.valueOf(resultInfo.get("Created"));
                String id = resultInfo.get("Id");
                String error = resultInfo.get("Error");
                if (success && created) {
                    System.out.println("Created row with id " + id);
                } else if (!success) {
                    System.out.println("Failed with error: " + error);
                }
            }
        }
    }
```

This code retrieves the results for each record and reports whether the operation succeeded or failed. If an error occurred for a record, the code prints out the error.

Complete Quick Start Sample

Now that you're more familiar with jobs and batches, you can copy and paste the entire quick start sample and use it:

```
import java.io.*;
import java.util.*;

import com.sforce.async.*;
import com.sforce.soap.partner.PartnerConnection;
import com.sforce.ws.ConnectionException;
import com.sforce.ws.ConnectorConfig;


public class BulkExample {


    public static void main(String[] args)
      throws AsyncApiException, ConnectionException, IOException {
        BulkExample example = new BulkExample();
        // Replace arguments below with your credentials and test file name
        // The first parameter indicates that we are loading Account records
       example.runSample("Account", "myUser@myOrg.com", "myPassword", "mySampleData.csv");

    }

    /**
     * Creates a Bulk API job and uploads batches for a CSV file.
     */
    public void runSample(String sobjectType, String userName,
              String password, String sampleFileName)
            throws AsyncApiException, ConnectionException, IOException {
        BulkConnection connection = getBulkConnection(userName, password);
        JobInfo job = createJob(sobjectType, connection);
        List<BatchInfo> batchInfoList = createBatchesFromCSVFile(connection, job,
            sampleFileName);
        closeJob(connection, job.getId());
        awaitCompletion(connection, job, batchInfoList);
        checkResults(connection, job, batchInfoList);
    }



    /**
     * Gets the results of the operation and checks for errors.
     */
    private void checkResults(BulkConnection connection, JobInfo job,
              List<BatchInfo> batchInfoList)
            throws AsyncApiException, IOException {
        // batchInfoList was populated when batches were created and submitted
        for (BatchInfo b : batchInfoList) {
            CSVReader rdr =
              new CSVReader(connection.getBatchResultStream(job.getId(), b.getId()));
            List<String> resultHeader = rdr.nextRecord();
            int resultCols = resultHeader.size();

            List<String> row;
            while ((row = rdr.nextRecord()) != null) {
                Map<String, String> resultInfo = new HashMap<String, String>();
                for (int i = 0; i < resultCols; i++) {
                    resultInfo.put(resultHeader.get(i), row.get(i));
                }
                boolean success = Boolean.valueOf(resultInfo.get("Success"));
                boolean created = Boolean.valueOf(resultInfo.get("Created"));
                String id = resultInfo.get("Id");
                String error = resultInfo.get("Error");
                if (success && created) {
                    System.out.println("Created row with id " + id);
                } else if (!success) {
                    System.out.println("Failed with error: " + error);
                }
            }
        }
    }



    private void closeJob(BulkConnection connection, String jobId)
          throws AsyncApiException {
        JobInfo job = new JobInfo();
        job.setId(jobId);
        job.setState(JobStateEnum.Closed);
        connection.updateJob(job);
    }



    /**
     * Wait for a job to complete by polling the Bulk API.
     *
     * @param connection
     *            BulkConnection used to check results.
     * @param job
     *            The job awaiting completion.
     * @param batchInfoList
     *            List of batches for this job.
     * @throws AsyncApiException
     */
    private void awaitCompletion(BulkConnection connection, JobInfo job,
          List<BatchInfo> batchInfoList)
            throws AsyncApiException {
        long sleepTime = 0L;
        Set<String> incomplete = new HashSet<String>();
        for (BatchInfo bi : batchInfoList) {
            incomplete.add(bi.getId());
        }
        while (!incomplete.isEmpty()) {
            try {
                Thread.sleep(sleepTime);
            } catch (InterruptedException e) {}
            System.out.println("Awaiting results..." + incomplete.size());
            sleepTime = 10000L;
            BatchInfo[] statusList =
              connection.getBatchInfoList(job.getId()).getBatchInfo();
            for (BatchInfo b : statusList) {
                if (b.getState() == BatchStateEnum.Completed
                  || b.getState() == BatchStateEnum.Failed) {
                    if (incomplete.remove(b.getId())) {
                        System.out.println("BATCH STATUS:\n" + b);
                    }
                }
            }
        }
    }



    /**
     * Create a new job using the Bulk API.
     *
     * @param sobjectType
     *            The object type being loaded, such as "Account"
     * @param connection
     *            BulkConnection used to create the new job.
     * @return The JobInfo for the new job.
     * @throws AsyncApiException
     */
    private JobInfo createJob(String sobjectType, BulkConnection connection)
          throws AsyncApiException {
        JobInfo job = new JobInfo();
        job.setObject(sobjectType);
        job.setOperation(OperationEnum.insert);
        job.setContentType(ContentType.CSV);
        job = connection.createJob(job);
        System.out.println(job);
        return job;
    }



    /**
     * Create the BulkConnection used to call Bulk API operations.
     */
    private BulkConnection getBulkConnection(String userName, String password)
          throws ConnectionException, AsyncApiException {
        ConnectorConfig partnerConfig = new ConnectorConfig();
        partnerConfig.setUsername(userName);
        partnerConfig.setPassword(password);
       partnerConfig.setAuthEndpoint("https://login.salesforce.com/services/Soap/u/55.0");

        // Creating the connection automatically handles login and stores
        // the session in partnerConfig
        new PartnerConnection(partnerConfig);
        // When PartnerConnection is instantiated, a login is implicitly
        // executed and, if successful,
        // a valid session is stored in the ConnectorConfig instance.
        // Use this key to initialize a BulkConnection:
        ConnectorConfig config = new ConnectorConfig();
        config.setSessionId(partnerConfig.getSessionId());
        // The endpoint for the Bulk API service is the same as for the normal
        // SOAP uri until the /Soap/ part. From here it's '/async/versionNumber'
        String soapEndpoint = partnerConfig.getServiceEndpoint();
        String apiVersion = "55.0";
        String restEndpoint = soapEndpoint.substring(0, soapEndpoint.indexOf("Soap/"))
            + "async/" + apiVersion;
        config.setRestEndpoint(restEndpoint);
        // This should only be false when doing debugging.
        config.setCompression(true);
        // Set this to true to see HTTP requests and responses on stdout
        config.setTraceMessage(false);
        BulkConnection connection = new BulkConnection(config);
        return connection;
    }



    /**
     * Create and upload batches using a CSV file.
     * The file into the appropriate size batch files.
     *
     * @param connection
     *            Connection to use for creating batches
     * @param jobInfo
     *            Job associated with new batches
     * @param csvFileName
     *            The source file for batch data
     */
    private List<BatchInfo> createBatchesFromCSVFile(BulkConnection connection,
          JobInfo jobInfo, String csvFileName)
            throws IOException, AsyncApiException {
        List<BatchInfo> batchInfos = new ArrayList<BatchInfo>();
        BufferedReader rdr = new BufferedReader(
            new InputStreamReader(new FileInputStream(csvFileName))
        );
        // read the CSV header row
        byte[] headerBytes = (rdr.readLine() + "\n").getBytes("UTF-8");
        int headerBytesLength = headerBytes.length;
        File tmpFile = File.createTempFile("bulkAPIInsert", ".csv");

        // Split the CSV file into multiple batches
        try {
            FileOutputStream tmpOut = new FileOutputStream(tmpFile);
            int maxBytesPerBatch = 10000000; // 10 million bytes per batch
            int maxRowsPerBatch = 10000; // 10 thousand rows per batch
            int currentBytes = 0;
            int currentLines = 0;
            String nextLine;
            while ((nextLine = rdr.readLine()) != null) {
                byte[] bytes = (nextLine + "\n").getBytes("UTF-8");
                // Create a new batch when our batch size limit is reached
                if (currentBytes + bytes.length > maxBytesPerBatch
                  || currentLines > maxRowsPerBatch) {
                    createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
                    currentBytes = 0;
                    currentLines = 0;
                }
                if (currentBytes == 0) {
                    tmpOut = new FileOutputStream(tmpFile);
                    tmpOut.write(headerBytes);
                    currentBytes = headerBytesLength;
                    currentLines = 1;
                }
                tmpOut.write(bytes);
                currentBytes += bytes.length;
                currentLines++;
            }
            // Finished processing all rows
            // Create a final batch for any remaining data
            if (currentLines > 1) {
                createBatch(tmpOut, tmpFile, batchInfos, connection, jobInfo);
            }
        } finally {
            tmpFile.delete();
        }
        return batchInfos;
    }

    /**
     * Create a batch by uploading the contents of the file.
     * This closes the output stream.
     *
     * @param tmpOut
     *            The output stream used to write the CSV data for a single batch.
     * @param tmpFile
     *            The file associated with the above stream.
     * @param batchInfos
     *            The batch info for the newly created batch is added to this list.
     * @param connection
     *            The BulkConnection used to create the new batch.
     * @param jobInfo
     *            The JobInfo associated with the new batch.
     */
    private void createBatch(FileOutputStream tmpOut, File tmpFile,
      List<BatchInfo> batchInfos, BulkConnection connection, JobInfo jobInfo)
              throws IOException, AsyncApiException {
        tmpOut.flush();
        tmpOut.close();
        FileInputStream tmpInputStream = new FileInputStream(tmpFile);
        try {
            BatchInfo batchInfo =
              connection.createBatchFromStream(jobInfo, tmpInputStream);
            System.out.println(batchInfo);
            batchInfos.add(batchInfo);

        } finally {
            tmpInputStream.close();
        }
    }


}
```

Map Data Fields

To use Bulk API to import data that was exported directly from Microsoft Outlook, Google Contacts, and other third-party sources, map data fields in any CSV import file to Salesforce data fields. It's not necessary for the CSV import file to be Bulk API-compatible.

For example, say you have a CSV import file that includes a field called `Number` that you want to map to the standard Salesforce field `AccountNumber`. When you add a batch job using Bulk API, data from your `Number` field is imported into (or updates) the `AccountNumber` field in Salesforce.

To add a batch job that maps data fields to Salesforce data fields:

1. Create a transformation spec (spec.csv) that defines data field mappings. (This file is different from the CSV import file that contains your data.)
2. Create a job that specifies an object and action, just as you would for any other Bulk API job.
3. Upload the transformation spec.
4. Send data to the server in batches.

Create a Transformation Spec That Defines Mappings

The transformation spec (spec.csv) provides the instructions for how to map the data in your import file to Salesforce data fields.

The spec.csv file contains four fields:

Field	Description
Salesforce Field	The Salesforce field you want to map to.
Csv Header	The field in your import file you want to map.
Value	A default value. Bulk API uses this value in two instances: • When there is no value present in the import file for the field specified in the Csv Header field • When there is no value defined for the Csv Header field in the spec.csv file This field is optional.
Hint	Tells Bulk API how to interpret data in the import file. Bulk API can use this value to do two things: • Interpret Java format strings for date and time fields • Define what is true using regular expressions for boolean fields This field is optional.

Here is a sample spec.csv file:

```
Salesforce Field,Csv Header,Value,Hint
Name,Full Name,,
Title,Job Title,,
LeadSource,Lead Source,Import,
Description,,Imported from XYZ.csv,
Birthdate,Date of Birth,,dd MM yy
```

This spec.csv file tells Bulk API to:

- Map the `Full Name` field in the import file to the `LastName` and `FirstName` fields in Salesforce.
- Map the `Job Title` field in the import file to the `Title` field in Salesforce.
- Map the `Lead Source` field in the import file to the `LeadSource` field in Salesforce, *and* use `Import as the default value` when no values are present are in the import file.
- Use `Imported from XYZ.csv` as the default value for the `Description` field in Salesforce.
- Map the `Date of Birth` field in the import file to the `Birthdate` field in Salesforce, *and* use the `dd MM yy` format to convert `Date of Birth` field formats into an acceptable format for Bulk API.

The corresponding contents of the import file can look like this:

```
Full Name,Job Title,Lead Source,Date of Birth,Comment
"Cat, Tom",DSH,Interview,10 Feb 40,likes Jerry
Jerry Mouse,House Mouse,,10 Feb 40,likes Tom
```

The corresponding request body after transformation looks like this:

```
LastName,FirstName,Title,LeadSource,Description,Birthdate
Cat,Tom,DSH,Interview,Imported from XYZ.csv,1940-02-10Z
Mouse,Jerry,House Mouse,Import,Imported from XYZ.csv,1940-02-10Z
```

Upload the Transformation Spec

To upload the transformation spec, send a POST request to this URI:

```
https://MyDomainName.my.salesforce.com/services/async/APIversion/job/jobid/spec
```

You can find the My Domain name and My Domain login URL for your org on the My Domain page in Setup.

Considerations

- Transformation specs must be CSV files. XML and JSON files are not supported.
- Transformation specs (spec.csv files) must use UTF-8 encoding. CSV import files do not need to use UTF-8 encoding. (You can specify the encoding in the Content-Type header.)
- Transformation specs are not persistent; their scopes are limited to the current job.

Bulk API End-of-Life

Salesforce is committed to supporting each API version for a minimum of three years from the date of first release. In order to mature and improve the quality and performance of the API, versions that are more than three years old might cease to be supported.

When an API version is to be deprecated, advance notice is given at least one year before support ends. Salesforce will directly notify customers using API versions planned for deprecation.

Note: Versions 16.0 through 20.0 of Bulk API have now been deprecated and are no longer supported. You can continue to access these legacy API versions until Summer '22 is released, at which point these legacy versions will be retired and will become unavailable. For more information, see Knowledge Article: Salesforce Platform API Versions 7.0 through 20.0 Retirement

Note: Versions 21.0 through 30.0 of Bulk API will be retired in the Summer '22 release. For more information, see Knowledge Article: Salesforce Platform API Versions 21.0 through 30.0 Retirement

GLOSSARY

A |B |C |D |E |F |G |H |I |J |K |L |M |N |O |P |Q |R |S |T |U |V |W |X |Y |Z

A

Apex

Apex is a strongly typed, object-oriented programming language that allows developers to execute flow and transaction control statements on the Lightning platform server in conjunction with calls to the Lightning Platform API. Using syntax that looks like Java and acts like database stored procedures, Apex enables developers to add business logic to most system events, including button clicks, related record updates, and Visualforce pages. Apex code can be initiated by Web service requests and from triggers on objects.

App

Short for "application." A collection of components such as tabs, reports, dashboards, and Visualforce pages that address a specific business need. Salesforce provides standard apps such as Sales and Service. You can customize the standard apps to match the way you work. In addition, you can package an app and upload it to the AppExchange along with related components such as custom fields, custom tabs, and custom objects. Then, you can make the app available to other Salesforce users from the AppExchange.

Application Programming Interface (API)

The interface that a computer system, library, or application provides to allow other computer programs to request services from it and exchange data.

Asynchronous Calls

A call that doesn't return results immediately because the operation can take a long time. Calls in the Metadata API and Bulk API 2.0 are asynchronous.

B

Batch, Bulk API

A batch is a CSV, XML, or JSON representation of a set of records in the Bulk API. You can process a set of records by creating a job that contains one or more batches. The server processes each batch independently, not necessarily in the order it is received. See Job, Bulk API.

Boolean Operators

You can use Boolean operators in report filters to specify the logical relationship between two values. For example, the AND operator between two values yields search results that include both values. Likewise, the OR operator between two values yields search results that include either value.

Bulk API 2.0

The REST-based Bulk API 2.0 is optimized for processing large sets of data. It allows you to query, insert, update, upsert, or delete a large number of records asynchronously by submitting a job that is processed in the background by Salesforce. See also SOAP API.

C

Client App

An app that runs outside the Salesforce user interface and uses only the Lightning Platform API or Bulk API 2.0. It typically runs on a desktop or mobile device. These apps treat the platform as a data source, using the development model of whatever tool and platform for which they are designed.

CSV (Comma Separated Values)

A file format that enables the sharing and transportation of structured data. The import wizards, Data Loader and the Bulk API 2.0 support CSV. Each line in a CSV file represents a record. A comma separates each field value in the record.

Custom Field

A field that can be added in addition to the standard fields to customize Salesforce for your organization's needs.

Custom Object

Custom records that allow you to store information unique to your organization.

D

Data Loader

A Lightning Platform tool used to import and export data from your Salesforce organization.

Database

An organized collection of information. The underlying architecture of the Lightning Platform includes a database where your data is stored.

Database Table

A list of information, presented with rows and columns, about the person, thing, or concept you want to track. See also Object.

Decimal Places

Parameter for number, currency, and percent custom fields that indicates the total number of digits you can enter to the right of a decimal point, for example, 4.98 for an entry of 2. Note that the system rounds the decimal numbers you enter, if necessary. For example, if you enter 4.986 in a field with `Decimal Places` of 2, the number rounds to 4.99. Salesforce uses the round half-up rounding algorithm. Half-way values are always rounded up. For example, 1.45 is rounded to 1.5. –1.45 is rounded to –1.5.

Dependent Field

Any custom picklist or multi-select picklist field that displays available values based on the value selected in its corresponding controlling field.

Developer Edition

A free, fully-functional Salesforce organization designed for developers to extend, integrate, and develop with the Lightning Platform. Developer Edition accounts are available on developer.salesforce.com.

Salesforce Developers

The Salesforce Developers website at developer.salesforce.com provides a full range of resources for platform developers, including sample code, toolkits, an online developer community, and the ability to obtain limited Lightning Platform environments.

E

Enterprise Edition

A Salesforce edition designed for larger, more complex businesses.

F

Field

A part of an object that holds a specific piece of information, such as a text or currency value.

Field-Level Security

Settings that determine whether fields are hidden, visible, read only, or editable for users. Available in Professional, Enterprise, Unlimited, Performance, and Developer Editions.

Lightning Platform

The Salesforce platform for building applications in the cloud. Lightning Platform combines a powerful user interface, operating system, and database to allow you to customize and deploy applications in the cloud for your entire enterprise.

Salesforce Extensions for Visual Studio Code

The Salesforce extension pack for Visual Studio Code includes tools for developing on the Salesforce platform in the lightweight, extensible VS Code editor. These tools provide features for working with development orgs (scratch orgs, sandboxes, and DE orgs), Apex, Aura components, and Visualforce.

Ant Migration Tool

A toolkit that allows you to write an Apache Ant build script for migrating Lightning Platform components between a local file system and a Salesforce organization.

Foreign Key

A field whose value is the same as the primary key of another table. You can think of a foreign key as a copy of a primary key from another table. A relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

Formula Field

A type of custom field. Formula fields automatically calculate their values based on the values of merge fields, expressions, or other values.

Function

Built-in formulas that you can customize with input parameters. For example, the DATE function creates a date field type from a given year, month, and day.

G

Gregorian Year

A calendar based on a 12-month structure used throughout much of the world.

H

No Glossary items for this entry.

I

ID

See Salesforce Record ID.

Instance

The cluster of software and hardware represented as a single logical server that hosts an organization's data and runs their applications. The Lightning Platform runs on multiple instances, but data for any single organization is always stored on a single instance.

Integration User

A Salesforce user defined solely for client apps or integrations. Also referred to as the logged-in user in a SOAP API context.

ISO Code

The International Organization for Standardization country code, which represents each country by two letters.

J

Job, Bulk API 2.0

A job in the Bulk API 2.0 specifies which object is being processed (for example, Account, Opportunity) and what type of action is being used (insert, upsert, update, or delete). You process a set of records by creating a job.

JSON (JavsScript Object Notation)

JSON is a lightweight format for transferring data.

K

No Glossary items for this entry.

L

Locale

The country or geographic region in which the user is located. The setting affects the format of date and number fields, for example, dates in the English (United States) locale display as 06/30/2000 and as 30/06/2000 in the English (United Kingdom) locale.

In Professional, Enterprise, Unlimited, Performance, and Developer Edition organizations, a user's individual `Locale` setting overrides the organization's `Default Locale` setting. In Personal and Group Editions, the organization-level locale field is called `Locale`, not `Default Locale`.

Logged-in User

In a SOAP API context, the username used to log into Salesforce. Client applications run with the permissions and sharing of the logged-in user. Also referred to as an integration user.

Lookup Field

A type of field that contains a linkable value to another record. You can display lookup fields on page layouts where the object has a lookup or master-detail relationship with another object. For example, cases have a lookup relationship with assets that allows users to select an asset using a lookup dialog from the case edit page and click the name of the asset from the case detail page.

M

Managed Package

A collection of application components that is posted as a unit on the AppExchange and associated with a namespace and possibly a License Management Organization. To support upgrades, a package must be managed. An organization can create a single managed package that can be downloaded and installed by many different organizations. Managed packages differ from unmanaged packages by having some locked components, allowing the managed package to be upgraded later. Unmanaged packages do not include locked components and cannot be upgraded. In addition, managed packages obfuscate certain components (like Apex) on subscribing organizations to protect the intellectual property of the developer.

Manual Sharing
Record-level access rules that allow record owners to give read and edit permissions to other users who might not have access to the record any other way.

Many-to-Many Relationship
A relationship where each side of the relationship can have many children on the other side. Many-to-many relationships are implemented through the use of junction objects.

Master-Detail Relationship
A relationship between two different types of records that associates the records with each other. For example, accounts have a master-detail relationship with opportunities. This type of relationship affects record deletion, security, and makes the lookup relationship field required on the page layout.

Metadata
Information about the structure, appearance, and functionality of an organization and any of its parts. Lightning Platform uses XML to describe metadata.

Multitenancy
An application model where all users and apps share a single, common infrastructure and code base.

N

Native App
An app that is built exclusively with setup (metadata) configuration on Lightning Platform. Native apps do not require any external services or infrastructure.

O

Object
An object allows you to store information in your Salesforce organization. The object is the overall definition of the type of information you are storing. For example, the case object allow you to store information regarding customer inquiries. For each object, your organization will have multiple records that store the information about specific instances of that type of data. For example, you might have a case record to store the information about Joe Smith's training inquiry and another case record to store the information about Mary Johnson's configuration issue.

Object-Level Security
Settings that allow an administrator to hide whole objects from users so that they don't know that type of data exists. Object-level security is specified with object permissions.

One-to-Many Relationship
A relationship in which a single object is related to many other objects. For example, an account may have one or more related contacts.

Organization-Wide Defaults
Settings that allow you to specify the baseline level of data access that a user has in your organization. For example, you can set organization-wide defaults so that any user can see any record of a particular object that is enabled via their object permissions, but they need extra permissions to edit one.

Outbound Message

An outbound message sends information to a designated endpoint, like an external service. Outbound messages are configured from Setup. You must configure the external endpoint and create a listener for the messages using the SOAP API.

Owner

Individual user to which a record (for example, a contact or case) is assigned.

P

Package

A group of Lightning Platform components and applications that are made available to other organizations through the AppExchange. You use packages to bundle an app along with any related components so that you can upload them to AppExchange together.

Parent Account

An organization or company that an account is affiliated. By specifying a parent for an account, you can get a global view of all parent/subsidiary relationships using the **View Hierarchy** link.

Picklist

Selection list of options available for specific fields in a Salesforce object, for example, the `Industry` field for accounts. Users can choose a single value from a list of options rather than make an entry directly in the field. See also Master Picklist.

Picklist (Multi-Select)

Selection list of options available for specific fields in a Salesforce object. Multi-select picklists allow users to choose one or more values. Users can choose a value by double clicking on it, or choose additional values from a scrolling list by holding down the CTRL key while clicking a value and using the arrow icon to move them to the selected box.

Picklist Values

Selections displayed in drop-down lists for particular fields. Some values come predefined, and other values can be changed or defined by an administrator.

Platform Edition

A Salesforce edition based on Enterprise, Unlimited, or Performance Edition that does not include any of the standard Salesforce apps, such as Sales or Service & Support.

Primary Key

A relational database concept. Each table in a relational database has a field in which the data value uniquely identifies the record. This field is called the primary key. The relationship is made between two tables by matching the values of the foreign key in one table with the values of the primary key in another.

Production Organization

A Salesforce organization that has live users accessing data.

Professional Edition

A Salesforce edition designed for businesses who need full-featured CRM functionality.

Q

Query String Parameter

A name-value pair that's included in a URL, typically after a '?' character. For example:

```
https://yourInstance.salesforce.com/001/e?name=value
```

R

Record

A single instance of a Salesforce object. For example, "John Jones" might be the name of a contact record.

Record Name

A standard field on all Salesforce objects. Whenever a record name is displayed in a Lightning Platform application, the value is represented as a link to a detail view of the record. A record name can be either free-form text or an autonumber field. `Record Name` does not have to be a unique value.

Record Type

A record type is a field available for certain records that can include some or all of the standard and custom picklist values for that record. You can associate record types with profiles to make only the included picklist values available to users with that profile.

Record-Level Security

A method of controlling data in which you can allow a particular user to view and edit an object, but then restrict the records that the user is allowed to see.

Recycle Bin

A page that lets you view and restore deleted information. Access the Recycle Bin either by using the link in the sidebar in Salesforce Classic or from the App Launcher in Lightning Experience.

Related Object

Objects chosen by an administrator to display in the Agent console's mini view when records of a particular type are shown in the console's detail view. For example, when a case is in the detail view, an administrator can choose to display an associated account, contact, or asset in the mini view.

Relationship

A connection between two objects, used to create related lists in page layouts and detail levels in reports. Matching values in a specified field in both objects are used to link related data; for example, if one object stores data about companies and another object stores data about people, a relationship allows you to find out which people work at the company.

Relationship Query

In a SOQL context, a query that traverses the relationships between objects to identify and return results. Parent-to-child and child-to-parent syntax differs in SOQL queries.

Role Hierarchy

A record-level security setting that defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.

Roll-Up Summary Field

A field type that automatically provides aggregate values from child records in a master-detail relationship.

Running User

Each dashboard has a *running user*, whose security settings determine which data to display in a dashboard. If the running user is a specific user, all dashboard viewers see data based on the security settings of that user—regardless of their own personal security settings. For dynamic dashboards, you can set the running user to be the logged-in user, so that each user sees the dashboard according to his or her own access level.

S

SaaS

See Software as a Service (SaaS).

Salesforce Record ID
A unique 15- or 18-character alphanumeric string that identifies a single record in Salesforce.

Salesforce SOA (Service-Oriented Architecture)
A powerful capability of Lightning Platform that allows you to make calls to external Web services from within Apex.

Sandbox
A nearly identical copy of a Salesforce production organization for development, testing, and training. The content and size of a sandbox varies depending on the type of sandbox and the edition of the production organization associated with the sandbox.

Session ID
An authentication token that is returned when a user successfully logs in to Salesforce. The Session ID prevents a user from having to log in again every time they want to perform another action in Salesforce. Different from a record ID or Salesforce ID, which are terms for the unique ID of a Salesforce record.

Session Timeout
The time after login before a user is automatically logged out. Sessions expire automatically after a predetermined length of inactivity, which can be configured in Salesforce from Setup by clicking **Security Controls**. The default is 120 minutes (two hours). The inactivity timer is reset to zero if a user takes an action in the web interface or makes an API call.

Setup
A menu where administrators can customize and define organization settings and Lightning Platform apps. Depending on your organization's user interface settings, Setup may be a link in the user interface header or in the dropdown list under your name.

Sharing
Allowing other users to view or edit information you own. There are different ways to share data:

- Sharing Model—defines the default organization-wide access levels that users have to each other's information and whether to use the hierarchies when determining access to data.
- Role Hierarchy—defines different levels of users such that users at higher levels can view and edit information owned by or shared with users beneath them in the role hierarchy, regardless of the organization-wide sharing model settings.
- Sharing Rules—allow an administrator to specify that all information created by users within a given group or role is automatically shared to the members of another group or role.
- Manual Sharing—allows individual users to share records with other users or groups.
- Apex-Managed Sharing—enables developers to programmatically manipulate sharing to support their application's behavior. See Apex-Managed Sharing.

Sharing Model
Behavior defined by your administrator that determines default access by users to different types of records.

Sharing Rule
Type of default sharing created by administrators. Allows users in a specified group or role to have access to all information created by users within a given group or role.

SOAP (Simple Object Access Protocol)
A protocol that defines a uniform way of passing XML-encoded data.

Software as a Service (SaaS)
A delivery model where a software application is hosted as a service and provided to customers via the Internet. The SaaS vendor takes responsibility for the daily maintenance, operation, and support of the application and each customer's data. The service alleviates the need for customers to install, configure, and maintain applications with their own hardware, software, and related IT resources. Services can be delivered using the SaaS model to any market segment.

SOQL (Salesforce Object Query Language)
A query language that allows you to construct simple but powerful query strings and to specify the criteria that selects data from the Lightning Platform database.

SOSL (Salesforce Object Search Language)

A query language that allows you to perform text-based searches using the Lightning Platform API.

Standard Object

A built-in object included with the Lightning Platform. You can also build custom objects to store information that is unique to your app.

T

Translation Workbench

The Translation Workbench lets you specify languages you want to translate, assign translators to languages, create translations for customizations you've made to your Salesforce organization, and override labels and translations from managed packages. Everything from custom picklist values to custom fields can be translated so your global users can use Salesforce in their language.

U

Unlimited Edition

Unlimited Edition is Salesforce's solution for maximizing your success and extending that success across the entire enterprise through the Lightning Platform.

Unmanaged Package

A package that cannot be upgraded or controlled by its developer.

V

Visualforce

A simple, tag-based markup language that allows developers to easily define custom pages and components for apps built on the platform. Each tag corresponds to a coarse or fine-grained component, such as a section of a page, a related list, or a field. The components can either be controlled by the same logic that is used in standard Salesforce pages, or developers can associate their own logic with a controller written in Apex.

W

Web Service

A mechanism by which two applications can easily exchange data over the Internet, even if they run on different platforms, are written in different languages, or are geographically remote from each other.

Web Services API

Term describing the original Salesforce Platform web services application programming interface (API) that provides access to your Salesforce org's information. See relevant developer guides for SOAP, REST, or Bulk APIs of interest.

WSDL (Web Services Description Language) File

An XML file that describes the format of messages you send and receive from a Web service. Your development environment's SOAP client uses the Salesforce Enterprise WSDL or Partner WSDL to communicate with Salesforce using the SOAP API.

X

XML (Extensible Markup Language)

A markup language that enables the sharing and transportation of structured data. All Lightning Platform components that are retrieved or deployed through the Metadata API are represented by XML definitions.

Y

No Glossary items for this entry.

Z

No Glossary items for this entry.

INDEX