Bulk API Developer Guide

Version 36.0, Spring '16

@salesforcedocs

Last updated: April 27, 2016

CONTENTS

CHAPTER 1 Introduction to Bulk API

The Bulk API provides a programmatic option to quickly load your org's data into Salesforce. To use the API requires basic familiarity with software development, web services, and the Salesforce user interface.

The functionality described is available only if your org has the Bulk API feature enabled. This feature is enabled by default for Performance, Unlimited, Enterprise, and Developer Editions.

When to Use Bulk API

Bulk API is based on REST principles and is optimized for loading or deleting large sets of data. You can use it to query, insert, update, upsert, or delete many records asynchronously by submitting batches. Salesforce processes batches in the background.

SOAP API, in contrast, is optimized for real-time client applications that update a few records at a time. SOAP API can be used for processing many records, but when the data sets contain hundreds of thousands of records, SOAP API is less practical. Bulk API is designed to make it simple to process data from a few thousand to millions of records.

The easiest way to use Bulk API is to enable it for processing records in Data Loader using CSV files. Using Data Loader avoids the need to write your own client application.

When to Use SOAP API

SOAP API provides a powerful, convenient, and simple SOAP-based Web services interface for interacting with Salesforce. You can use SOAP API to create, retrieve, update, or delete records. You can also use SOAP API to perform searches and much more. Use SOAP API in any language that supports Web services.

For example, you can use SOAP API to integrate Salesforce with your organization's ERP and finance systems. You can also deliver real-time sales and support information to company portals, and populate critical business systems with customer information.

When to Use REST API

REST API provides a powerful, convenient, and simple REST-based Web services interface for interacting with Salesforce. Its advantages include ease of integration and development, and it's an excellent choice of technology for use with mobile applications and Web projects. However, if you have many records to process, consider using Bulk API, which is based on REST principles and optimized for large sets of data.

When to Use Metadata API

Use Metadata API to retrieve, deploy, create, update, or delete customizations for your organization. The most common use is to migrate changes from a sandbox or testing organization to your production environment. Metadata API is intended for managing customizations and for building tools that can manage the metadata model, not the data itself.

The easiest way to access the functionality in Metadata API is to use the Force.com IDE or Force.com Migration Tool. Both tools are built on top of Metadata API and use the standard Eclipse and Ant tools respectively to simplify working with Metadata API.

- Force.com IDE is built on the Eclipse platform, for programmers familiar with integrated development environments. Code, compile, test, and deploy from within the IDE.
- The Force.com Migration Tool is ideal if you use a script or the command line for moving metadata between a local directory and a Salesforce organization.

What You Can Do with Bulk API

The REST Bulk API lets you query, insert, update, upsert, or delete a large number of records asynchronously. The records can include binary attachments, such as Attachment objects or Salesforce CRM Content. You first send a number of batches to the server using an HTTP POST call and then the server processes the batches in the background. While batches are being processed, you can track progress by checking the status of the job using an HTTP GET call. All operations use HTTP GET or POST methods to send and receive CSV, XML, or JSON data.

Important: Currently base64 fields are not supported in queries with the 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 action is being used (query, insert, upsert, update, or delete). 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 is received. Batches may 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 the following steps.

1. Create a new job that specifies the object and action.
2. Send data to the server in a number of batches.
3. Once all data has been submitted, close the job. Once 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 either 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.

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

For information about using Data Loader to process CSV files, see the *Data Loader Guide*.

SEE ALSO:

- Data Loader Guide
- SOAP API Developer Guide
- Metadata API Developer's Guide
- Work with Jobs
- Work with Batches

CHAPTER 2 Quick Start

In this chapter ...

Use this sample to create HTTP requests that insert new contact records using the REST-based Bulk API. The instructions progress through logging in, submitting the records, checking status, and retrieving the results.

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 beginning this quick start. You may also wish to review the rest of this document to familiarize yourself with terms and concepts.

Set Up a Salesforce Developer Edition Organization

First, you must obtain a Salesforce Developer Edition organization and enable Bulk API.

1. Obtain a Salesforce Developer Edition organization.

   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. Even if you already have an Enterprise Edition, Unlimited Edition, or Performance Edition account, it's strongly recommended that you use Developer Edition for developing, staging, and testing your solutions against sample data to protect your organization's live data. This is especially true for applications that query, insert, update, or delete data (as opposed to simply reading data).

2. Enable Bulk API.

   You must have the "API Enabled" permission. This permission is enabled in the System Administrator profile.

Set Up Your Client Application

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 is not supported by Salesforce.

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

When running the cURL examples for the REST resources, you may 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, do one of the following:

- 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://instance_name.salesforce.com/services/data/v36.0/
-H "Authorization: Bearer
00D50000000IehZ\!AQcAQH0dMHZfz972Szmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1E6
LYUfiDUkWe6H34r1AAwOR8B8fLEz6n04NPGRrq0FM"
```

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

```
curl https://instance_name.salesforce.com/services/data/v36.0/
-H 'Authorization: Bearer sessionID'
```

Send HTTP Requests with cURL

Now that you have configured cURL, 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 the following format:

`Web_Services_SOAP_endpoint_instance_name``/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 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 85.

1. 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.
2. 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, 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.
3. 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.
4. Step 4: Close the Job
   When you're finished submitting batches to Salesforce, close the job. This informs Salesforce that no more batches will be submitted for the job, which, in turn, allows the monitoring page in Salesforce to return more meaningful statistics on the progress of the job.
5. Step 5: Check Batch Status
   You can check the status of an individual batch by running the following cURL command.
6. Step 6: Retrieve Batch Results
   Once a batch is `Completed`, you need to 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.

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

```
<?xml version="1.0" encoding="utf-8" ?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <n1:login xmlns:n1="urn:partner.soap.sforce.com">
      <n1:username>your_username</n1:username>
      <n1:password>your_password</n1:password>
    </n1:login>
  </env:Body>
</env:Envelope>
```

2. Replace *your_username* and *your_password* with your Salesforce user name and password.

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

   ```
   curl https://login.salesforce.com/services/Soap/u/36.0 -H "Content-Type: text/xml;
   charset=UTF-8" -H "SOAPAction: login" -d @login.txt
   ```

   The `Soap/u/` portion of the URI specifies the partner WSDL. You can use `Soap/c/` to specify the enterprise WSDL.

4. Salesforce 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 `na1-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, 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 the following 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 be all lower case. 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/36.0/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.

   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>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>36.0</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 24. For information about batch size limitations, see Batch size and limits on page 82.

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'll 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 the following 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 the following cURL command.

`curl https://`***`instance`***`.salesforce.com/services/async/36.0/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 the following.

```
<?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 does not 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

Bulk API Limits

Step 4: Close the Job

When you're finished submitting batches to Salesforce, close the job. This informs Salesforce that no more batches will be submitted for the job, which, in turn, 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/36.0/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 the following cURL command.

```
curl https://instance.salesforce.com/services/async/36.0/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 the following:

```
<?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. You need to retrieve the batch result to see the status of individual records.

You don't have to check the status of each batch individually. You can check the status for all batches that are part of the job by running the following cURL command:

```
curl https://instance.salesforce.com/services/async/36.0/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

Once a batch is `Completed`, you need to 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/36.0/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 the following:

```
"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

CHAPTER 3 Plan Bulk Data Loads

In this chapter ...

In most circumstances, the Bulk API is significantly faster than the SOAP-based API for loading large numbers of records. However, 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

These are some tips for planning your data loads for optimal processing time. Always test your data loads in a sandbox organization first. Note that the processing times may 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. To get batch results to see which records, if any, were processed, see Get Batch Results on page 48. 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

The following 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 possible, loading times will 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, you should 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 have to 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 42.

Batch sizes should be adjusted based on processing times. Start with 5000 records and adjust the batch size based on processing time. If it takes more than five minutes to process a batch, it may be beneficial to reduce the batch size. If it takes a few seconds, the batch size should be increased. If you get a timeout error when processing a batch, split your batch into smaller batches, and try again. For more information, see Bulk API Limits on page 81.

Note: For Bulk queries, the batch size is not applied to either the query result set, or the retrieved data size. If your bulk query is taking too long to process, you will need to 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, as well as 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 additional 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.

CHAPTER 4 Prepare Data Files

In this chapter ...

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 24.

For information about loading data from third-party sources, see Map Data Fields on page 99.

SEE ALSO:

Data Loader Guide

Find Field Names

After you have 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's 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 an object's field name:

1. From the object's management settings, go to the fields area.
2. Click the `Field Label` for a field.

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

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

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

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

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

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 18. 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 99.

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

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

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

Note the following when referencing relationships in CSV header rows:

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

Relationship Fields for Custom Objects

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

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

The following CSV file uses a relationship:

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

Relationships for Polymorphic Fields

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

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

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

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

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 following 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 this file to update existing accounts, any fields that aren't defined in the CSV file are ignored during the update. You must include all required fields when you create a record.

```
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."
```

Note that the `Description` field for the last record includes a line break, so 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 `Reports To` field for a contact is a reference to another contact.

You can add a reference to a related object for a field in an XML or JSON record by representing the relationship using the following syntax, where *`RelationshipName`* is the relationship name of the field and *`IndexedFieldName`* is the indexed field name that uniquely identifies the parent record:

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

Use the `describeSObjects()` call in the SOAP-based SOAP API to get the `relationshipName` property value for a field. You must 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`.

The following sample includes a contact record that includes the `Reports To` field, which is a reference to another contact. `ReportsTo` is the `relationshipName` property value for the `Reports To` field. In this case, the parent object for the `Reports To` 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, see the Field Properties column in the field table for each standard object.

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
   <sObject>
      <FirstName>Ray</Name>
      <LastName>Riordan</Description>
      <ReportsTo>
        <sObject>
          <Email>rwilliams@salesforcesample.com</Email>
        </sObject>
      </ReportsTo>
   </sObject>
</sObjects>
```

When using relationships in XML or JSON 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.

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 `Mother_Of_Child__r` `relationshipName` property for the field to indicate that you're referencing a relationship. Use the parent object's `External ID` field to uniquely identify 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 XML file shows usage of the relationship:

```
<?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 XML sample shows 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.

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

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.

```
<?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>`***`ObjectTypeName`***`</type>` element is only required for a polymorphic field. You get an error if you omit this element for a polymorphic field. You also get an error if you include this syntax for a field that is not polymorphic.

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 the following 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 the following 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. The batch fails if you do otherwise. 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`. In JSON, simply 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.

Sample XML File

The following 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

The following 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.

The following 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

CHAPTER 5 Load Binary Attachments

In this chapter ...

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.

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 the following 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

The following 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

The following 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

The following 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"
  }
]
```

SEE ALSO:

Create a Zip Batch File with Binary Attachments

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.

SEE ALSO:

Create a request.txt File

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 Send HTTP Requests with cURL on page 5.

1. Create a text file called `job.txt` containing the following 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 the following cURL command

   ```
   curl https://instance.salesforce.com/services/async/36.0/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 6.

   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>36.0</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 Batch with Binary Attachments

Create a Job

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 13.

1. Create a zip batch file. For this example, the file is named `request.zip`.

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

   ```
   curl https://instance.salesforce.com/services/async/36.0/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 the following.

   ```
   <?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 does not 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 Quick Start.

SEE ALSO:

Create a Job for Batches with Binary Attachments

Add a Batch to a Job

CHAPTER 6 Request Basics

In this chapter ...

Here are some basics about Bulk API requests, 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.

The URI where you send HTTP requests has the following format:

`Web_Services_SOAP_endpoint_instance_name/services/async/APIversion/Resource_address`

Think of the part of the URI through the API version as a base URI which is used for all operations. The part after the API version (`Resource_address`) varies depending on the job or batch being processed. For example, if your organization is on the `na5` instance and you're working with version 36.0 of Bulk API, your base URI would be `https://na5.salesforce.com/services/async/36.0`.

The instance name for your organization is returned in the LoginResult `serverUrl` field.

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.

The following example shows how to specify the required information once you have obtained it from the `login()` call.

```
POST /service/async/36.0/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: na5.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

Sample Client Application Using Java

CHAPTER 7 Work with Jobs

In this chapter ...

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 action is being used (query, insert, upsert, update, or delete).

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.

Create a Job

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

URI

`https://instance_name—api.salesforce.com/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 on page 71.

Warning: The operation value must be all lower case. 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
- Bulk API Limits
- About URIs
- JobInfo
- Quick Start

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, from Setup, enter `Bulk Data Load Jobs` in the `Quick Find` box, then select **Bulk Data Load Jobs**.

For more information, see “Monitoring 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
- Bulk API Limits
- Data Loader Guide

Close a Job

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

URI

`https://`***`instance_name`***`—api.salesforce.com/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

Bulk API Limits

About URIs

JobInfo

Quick Start

Get Job Details

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

URI

`https://`***`instance_name`***`—api.salesforce.com/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
- Bulk API Limits
- About URIs
- JobInfo
- Quick Start

Abort a Job

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

URI

`https://`***`instance_name`***`—api.salesforce.com/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
- Bulk API Limits
- About URIs
- JobInfo

Job and Batch Lifespan

All jobs and batches older than seven days are deleted.

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

For more information about limits, see Bulk API Limits on page 81.

SEE ALSO:

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

CHAPTER 8 Work with Batches

In this chapter ...

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 is 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 may 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.

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

Add a Batch to a Job

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

URI

`https://instance_name—api.salesforce.com/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 71.

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 99.

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

Bulk API Limits

About URIs

BatchInfo

Quick Start

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, enter *`Bulk Data Load Jobs`* in the `Quick Find` box, 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 more information, see “Monitoring 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
- Bulk API Limits
- About URIs
- BatchInfo
- Quick Start

Get Information for a Batch

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

URI

`https://`***`instance_name`***`—api.salesforce.com/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
- Bulk API Limits
- BatchInfo
- About URIs
- Work with Jobs
- Quick Start

Get Information for All Batches in a Job

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

URI

`https://instance_name—api.salesforce.com/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

Bulk API Limits

BatchInfo

About URIs

Work with Jobs

Quick Start

Interpret Batch State

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

`Queued`
: Processing of the batch has not started yet. If the job associated with this batch is aborted, the batch isn't processed and its state is set to `Not Processed`.

`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 have failed. A batch can be completed even if some or all the records have 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 been 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.

`Not Processed`
: 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
- Bulk API Limits
- BatchInfo
- About URIs
- Work with Jobs
- Quick Start

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, enter `Bulk Data Load Jobs` in the `Quick Find` box, 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

```
https://instance_name—api.salesforce.com/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

Bulk API Limits

About URIs

BatchInfo

Quick Start

Get Batch Results

Get results of a batch that has completed processing by sending a GET request to the following 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, enter `Bulk Data Load Jobs` in the `Quick Find` box, 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. The **View Response** link returns the same results as the following URI resource.

URI

`https://`***`instance_name`***`-api.salesforce.com/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 50.

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
Bulk API Limits
BatchInfo
About URIs
Work with Jobs
Quick Start

Handle Failed Records in Batches

A batch can have a `Completed` state even if some or all of the records have 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 been 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 was not 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 43 and Interpret Batch State on page 46.
2. Get the batch results.

   The following 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 was not 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 have examined each result record, you can manually fix each record in the error file and submit these records in a new batch. Repeat the earlier steps to check that each record is processed successfully.

SEE ALSO:

Add a Batch to a Job

Errors

Bulk API Limits

CHAPTER 9 Bulk Query

In this chapter ...

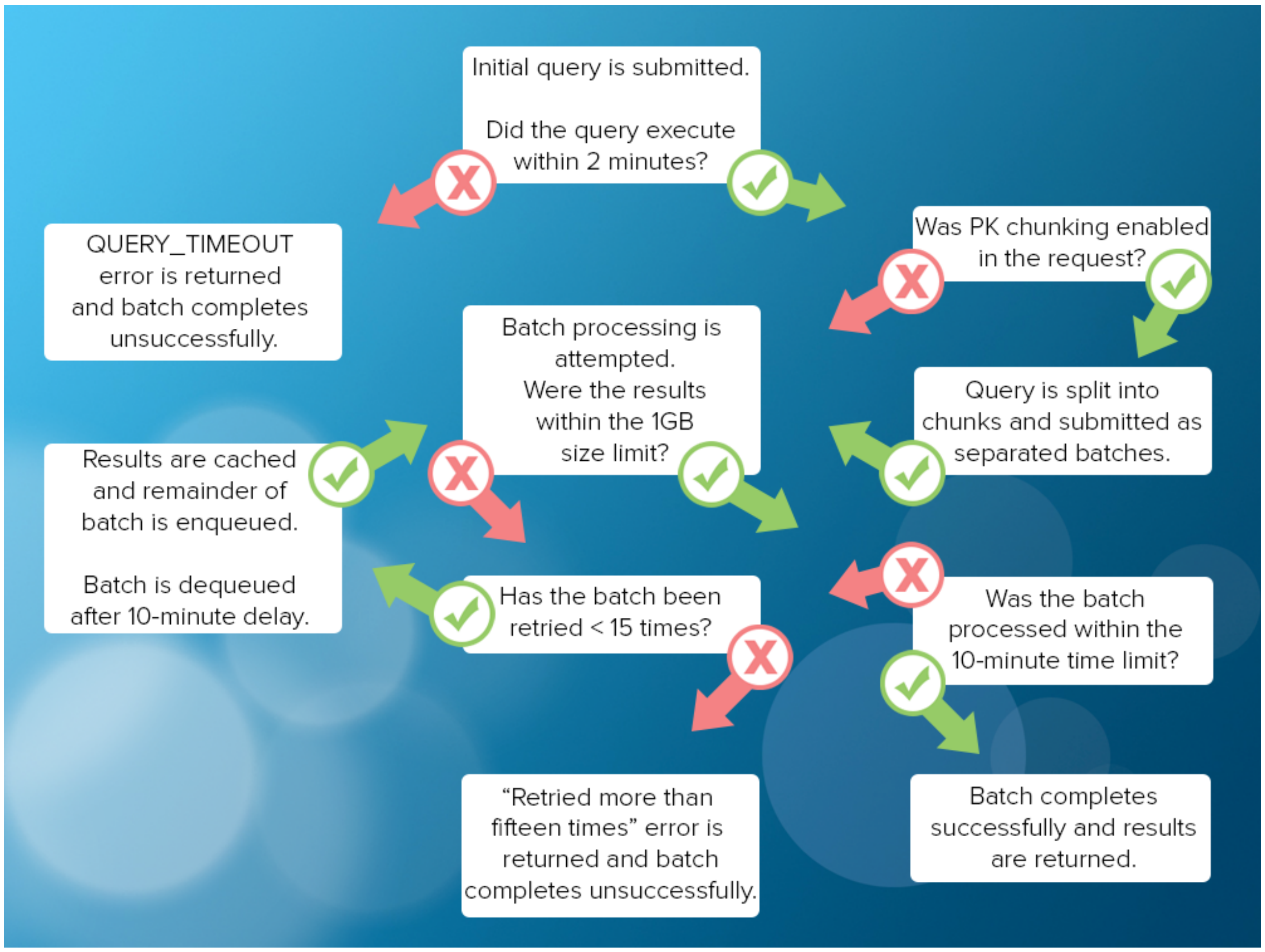
Use bulk query to efficiently query large data sets and reduce the number of API requests. A bulk query can retrieve up to 15 GB of data, divided into 15 1-GB files. The data formats supported are CSV, XML, and JSON.

How Bulk Queries Are Processed

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 seven days.

The following 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 adding a batch to a bulk query job, the Content-Type in the header for the request must be either `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

`https://`***`instance_name`***`—api.salesforce.com/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]
```

Note: Bulk API query doesn't support the following SOQL:

- COUNT
- ROLLUP
- SUM
- GROUP BY CUBE
- OFFSET
- Nested SOQL queries
- Relationship fields

Also, Bulk API can't access or query compound address or compound geolocation fields.

Example: **Requests, and Responses**

The following 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"
```

Example: **Java Example Using WSC**

The following 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(i==0 ? 30 * 1000 : 30 * 1000); //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 the following 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 the following 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/36.0/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 the following.

```
<?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 the following 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/36.0/job/jobId/batch
   ```

   `jobId` is the job ID returned in the response to the job createion.

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

   ```
   <?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 adding a batch to a bulk query job, the Content-Type in the header for the request must be either `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 the following cURL command to check the job status.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/job/jobId
   ```

   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>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 the following cURL command to check the batch status.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/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 the following.

   ```
   <?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 the following cURL command to retrieve the batch result list.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/job/jobId/batch/batchId/result
   ```

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

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <result-list xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <result>752x00000004CJE</result>
   </result-list>
   ```

2. Using the command-line window, execute the following cURL command to retrieve the results of the query.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/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 the following.

   ```
   "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 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 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/36.0/job`

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>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 the following 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 the following 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/36.0/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 the following.

```
<?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 the following 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/36.0/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 the following.

```
<?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 adding a batch to a bulk query job, the Content-Type in the header for the request must be either `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 the following cURL command to check the job status.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/job/jobId
   ```

   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>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 the following cURL command to check the status of the original batch.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/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 the following.

   ```
   <?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 the following cURL command to retrieve the remaining batches.

```
curl -H "X-SFDC-Session: sessionId"
https://instance.salesforce.com/services/async/36.0/job/jobId/batch
```

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

```
<?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 the following steps for each remaining batch.

1. Using the command-line window, execute the following cURL command to retrieve the batch result list.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/job/jobId/batch/batchId/result
   ```

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

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <result-list xmlns="http://www.force.com/2009/06/asyncapi/dataload">
     <result>752x00000004CJE</result>
   </result-list>
   ```

2. Using the command-line window, execute the following cURL command to retrieve the results of the query.

   ```
   curl -H "X-SFDC-Session: sessionId"
   https://instance.salesforce.com/services/async/36.0/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 the following.

```
"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 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 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/36.0/job`

   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>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

CHAPTER 10 Reference

In this chapter ...

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

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 the following URI:

`Web_Services_SOAP_endpoint_instance_name/services/async/APIversion/AsyncApi.xsd`

For example, if your organization is on the `na5` instance and you're working with version 36.0 of the Bulk API, the URI is:

`https://na5.salesforce.com/services/async/36.0/AsyncApi.xsd`

The instance name for your organization is returned in the LoginResult `serverUrl` field.

Schema and API Versions

The schema file is available for API versions prior to the current release. You can download the schema file for API version 18.0 and later. For example, if your organization is on the `na2` instance and you want to download the schema file for API version 18.0, use the following URI:

`https://na2.salesforce.com/services/async/18.0/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.

Name	Type	Request	Description
`apiActiveProcessingTime`	long	Do not specify for new job.	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. 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.
`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 may fail. If you're experiencing this issue, submit the job with `serial` concurrency mode. This guarantees that batches are processed one at a time. Note that using this option may significantly increase the processing time for a job.
`contentType`	ContentType		The content type for the job. The valid values are: • `CSV`—data in CSV format • `JSON`—data in JSON format • `XML`—data in XML format (default option) • `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()`.

Name	Type	Request	Description
`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.
`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` • `insert` • `query` • `upsert` • `update` • `hardDelete`

Name	Type	Request	Description
			Warning: The operation value must be all lower case. 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. 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 batches can be added to the job. • `Closed`: No new batches can be added to this job. Batches 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.

Name	Type	Request	Description
`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

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.

Name	Type	Request	Description
`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 has not started yet. If the job associated with this batch is aborted, the batch isn't processed and its state is set to `Not Processed`. • `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 have failed. A batch can be completed even if some or all the records have 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 been 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. • `Not Processed`: 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	Contains details about the state. For example, if 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 been 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`.

Name	Type	Request	Description
`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 75.

SEE ALSO:

- Work with Batches
- Interpret Batch State
- Quick Start
- SOAP API Developer Guide

Headers

These are 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.

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 won't be retried.
- `FALSE`. Unfinished batches in this job will be 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 PK Chunking request header to enable automatic primary key (PK) chunking for a bulk query job. PK chunking splits bulk queries on very 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 is supported for the following objects: Account, Campaign, CampaignMember, Case, Contact, Lead, LoginHistory, Opportunity, Task, User, and custom objects.

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 the following 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. 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.

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.

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.

- `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.
- `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
```

HTTP Status Codes

Operations that you perform with Bulk API return an HTTP status code. The following list shows the most common status codes and the Bulk API action that may have triggered them.

HTTP 200
The operation completed successfully.

HTTP 400
The operation failed to complete successfully due to an invalid request.

HTTP 405
An HTTP method other than GET or POST was sent to the URI.

HTTP 415
You may have set compression to an unsupported value. The only valid compression value is `gzip`. Compression is optional, but strongly recommended.

HTTP 500
Generally, a server error.

Errors

Operations that you perform with Bulk API might trigger error codes. The following list shows the most common error codes and the Bulk API action that might have 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 is not 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 the following 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 Binary content on page 82.

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 on page 82. 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 Guidelines for Data Loads on page 13.

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's Guide*.

SEE ALSO:

HTTP Status Codes

Handle Failed Records in Batches

Bulk API Limits

Note the following Bulk API limits.

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 is not processed and `stateMessage` is updated. Use the enterprise WSDL for the correct format for object records.

Batch limit

You can submit up to 5,000 batches per rolling 24 hour period. You can't create new batches associated with a job that is more than 24 hours old.

Batch lifespan

Batches and jobs that are older than seven days are removed from the queue regardless of 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 new 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 can contain a maximum of 10,000 records.
- A batch can contain a maximum of 10,000,000 characters for all the data in a batch.
- A field can contain a maximum of 32,000 characters.
- A record can contain a maximum of 5,000 fields.
- A record can contain a maximum of 400,000 characters for all its fields.
- A batch must contain some content or an error occurs.

For binary content limits, see Binary content on page 82.

Batch processing time

There is a five-minute limit for processing 100 records. Also, if it takes longer than 10 minutes to process a 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. To get batch results to see which records, if any, were processed, see Get Batch Results on page 48. If you get a timeout error when processing a batch, split your batch into smaller batches, and try again.

Note: For bulk queries, there is an additional two minute limit for processing the actual query which is separate from the batch processing limit. See `query` Limits.

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.

Compression

The only valid compression value is `gzip`. Compression is optional, but strongly recommended. Note that 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 contains records of one object. Optionally, the job may 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 seven days are removed from the queue regardless of 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 new 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

After a job has completed, the job status and batch result sets are available for 7 days after which this 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.

`query` Limits

Bulk API query has the following limitations:

Feature	Functionality
Retrieved file size	1 gigabyte.
Number of retrieved files	15 files. If the query needs to return more than 15 files, the query should be filtered to return less data. Bulk batch sizes are not used for bulk queries.
Number of attempts to query	15 attempts at 10 minutes each to process the batch. There is also a two 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 two minutes to process, a QUERY_TIMEOUT error is returned.
Length of time results are kept	7 days.

Bulk API query doesn't support the following SOQL:

- COUNT
- ROLLUP
- SUM

- GROUP BY CUBE
- OFFSET
- Nested SOQL queries
- Relationship fields

APPENDIX A 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 for you, to make copying and pasting easier.

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 have 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.

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`
2. Click the "Available versions" link that matches the API version you are using.
3. Click **Download (JAR)** and save the file to a local directory.

The Bulk API does not 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 "Restrict Where and When Users Can Log In To Salesforce" in the Salesforce online help.
2. From Setup, enter "API" in the `Quick Find` box, 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\wsc.jar com.sforce.ws.tools.wsdlc pathToWSDL\wsdlFilename
   .\wsdlGenFiles.jar
   ```

   For example, if `wsc.jar` is installed in `C:\salesforce\wsc`, and the partner WSDL is saved to `C:\salesforce\wsdl\partner`:

   ```
   java -classpath C:\salesforce\wsc\wsc.jar com.sforce.ws.tools.wsdlc
   C:\salesforce\wsdl\partner\partner.wsdl .\partner.jar
   ```

   `wsc.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 have 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.

The following 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'll 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

The following 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/36.0");

        // 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 = "36.0";
        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. The following 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 Bulk API Limits on page 81.

The processing of each batch comes with an overhead. Batch sizes should be large enough to minimize the overhead processing cost and small enough to be easily handled and transferred. Batch sizes between 1,000 and 10,000 records are considered reasonable.

The following 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 have been 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. A batch may take some time to complete depending on the size of the data set. During processing, the status of all batches can be retrieved and checked to see when they have completed.

```
    /**
     * 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

After all batches have completed, the results of each batch can be retrieved. Results should be retrieved whether the batch succeeded or failed, or even when 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. This can be achieved by keeping the original list of batches from when they were created and using this list to retrieve results, as shown in the following 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/36.0");

        // 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 = "36.0";
        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();
        }
    }
```

```
}
```

APPENDIX B 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. The CSV import file does not need to be Bulk API–compatible.

For example, you might 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 is different from the CSV import file that contains your data.)
2. Create a new 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

Field	Description
	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 might 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://instance_name—api.salesforce.com/services/async/APIversion/job/jobid/spec
```

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.

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 Force.com platform server in conjunction with calls to the Force.com 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 Call Center. 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 does not return results immediately because the operation may take a long time. Calls in the Metadata API and Bulk API 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 process a set of records by creating a job that contains one or more batches. Each batch is processed independently by the server, 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

The REST-based Bulk API 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 number of batches which are 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 Force.com API or Bulk API. 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 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 Force.com platform tool used to import and export data from your Salesforce organization.

Database

An organized collection of information. The underlying architecture of the Force.com 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 Force.com 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 Force.com 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.

Force.com
The Salesforce platform for building applications in the cloud. Force.com combines a powerful user interface, operating system, and database to allow you to customize and deploy applications in the cloud for your entire enterprise.

Force.com IDE
An Eclipse plug-in that allows developers to manage, author, debug and deploy Force.com applications in the Eclipse development environment.

Force.com Migration Tool
A toolkit that allows you to write an Apache Ant build script for migrating Force.com 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 Force.com 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

A job in the Bulk API 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 that contains one or more batches. See Batch, Bulk API.

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. Force.com 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 Force.com. 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 is a workflow, approval, or milestone action that sends the information you specify to an endpoint you designate, such as an external service. Outbound messaging is configured in the Salesforce setup menu. Then you must configure the external endpoint. You can 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 Force.com 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:

```
http://na1.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 Force.com 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 by using the link in the sidebar.

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 Force.com 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 editioin 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 he or she wants 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 period of 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 Force.com apps. Depending on your organization's user interface settings, Setup may be a link in the user interface header or in the drop-down 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 should be used to select data from the Force.com database.

SOSL (Salesforce Object Search Language)

A query language that allows you to perform text-based searches using the Force.com API.

Standard Object

A built-in object included with the Force.com 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 all of 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 Force.com 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

A Web services application programming interface that provides access to your Salesforce organization's information. See also SOAP API and Bulk API.

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 Force.com 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