



Salesforce: Summer '12

Database.com Bulk API Developer's Guide



Last updated: April 16 2012

© Copyright 2000–2012 salesforce.com, inc. All rights reserved. Salesforce.com is a registered trademark of salesforce.com, inc., as are other names and marks. Other marks appearing herein may be trademarks of their respective owners.

Table of Contents

Chapter 1: Introduction.....	3
Chapter 2: Quick Start.....	5
Setting Up Bulk API Permissions.....	5
Setting Up Your Client Application.....	5
Sending HTTP Requests with cURL.....	6
Step 1: Logging In Using the SOAP SOAP API.....	6
Step 2: Creating a Job.....	7
Step 3: Adding a Batch to the Job.....	8
Step 4: Closing the Job.....	9
Step 5: Checking Batch Status.....	10
Step 6: Retrieving Batch Results.....	10
Chapter 3: Planning Bulk Data Loads.....	12
General Guidelines for Data Loads.....	13
Chapter 4: Preparing Data Files.....	15
Finding Field Names.....	16
Valid Date Format in Records.....	16
Preparing CSV Files.....	16
Relationship Fields in a Header Row.....	17
Valid CSV Record Rows.....	18
Sample CSV File.....	18
Preparing XML Files.....	19
Relationship Fields in Records.....	19
Valid XML Records.....	21
Sample XML File.....	21
Chapter 5: Request Basics.....	23
About URIs.....	24
Setting a Session Header.....	24
Chapter 6: Working with Jobs.....	25
Creating a New Job.....	26
Monitoring a Job.....	27
Closing a Job.....	27
Getting Job Details.....	28
Aborting a Job.....	29
Job and Batch Lifespan.....	30
Chapter 7: Working with Batches.....	31
Adding a Batch to a Job.....	32
Monitoring a Batch.....	33

Getting Information for a Batch.....	34
Getting Information for All Batches in a Job.....	35
Interpreting Batch State.....	36
Getting a Batch Request.....	37
Getting Batch Results.....	38
Handling Failed Records in Batches.....	39
Chapter 8: Bulk Query.....	41
Bulk Query Details.....	42
Chapter 9: Reference.....	46
Schema.....	47
JobInfo.....	47
BatchInfo.....	50
HTTP Status Codes.....	52
Errors.....	53
Bulk API Limits.....	54
Appendix A: Sample Client Application Using Java.....	57
Set Up Your Client Application.....	57
Walk Through the Sample Code.....	58
Appendix B: Sample Bulk Query Using cURL.....	69
Walk Through the Bulk Query Sample.....	69
Glossary.....	73
Index.....	82

Chapter 1

Introduction

The Bulk API provides programmatic access to allow you to quickly load your organization's data into Database.com. To use this document, you should have a basic familiarity with software development, Web services, and the Database.com user interface.

Any functionality described in this guide is available only if your organization has the Bulk API feature enabled.

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 a large number of records asynchronously by submitting batches which are processed in the background by Database.com.

SOAP API, in contrast, is optimized for real-time client applications that update small numbers of records at a time. Although SOAP API can also be used for processing large numbers of records, when the data sets contain hundreds of thousands of records, it becomes 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. This 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 Database.com. 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.

When to Use REST API

REST API provides a powerful, convenient, and simple REST-based Web services interface for interacting with Database.com. 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 a large number of records to process, you may wish to use [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 test database 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. These tools are built on top of Metadata API and use the standard Eclipse and Ant tools respectively to simplify the task of working with Metadata API. Built on the Eclipse platform, the Force.com IDE provides a comfortable environment for programmers familiar with integrated development environments, allowing you to code, compile, test, and deploy all from within the IDE itself. The Force.com Migration Tool is ideal if you want to use a script or a command-line utility for moving metadata between a local directory and a Database.com organization.

What You Can Do with the Bulk API

The REST Bulk API lets you query, insert, update, upsert, or delete a large number of records asynchronously. 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 XML or CSV data.



Important: Currently base64 fields are not supported in queries with the Bulk API.

How the 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 or XML representation of a set of records and any references to binary attachments in an HTTP POST request. Once 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](#).

Chapter 2

Quick Start

Use the quick start sample in this section 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.

Setting Up Bulk API Permissions

First, you must enable Bulk API. To do so, you must have the “API Enabled” permission. This permission is enabled in the System Administrator profile.

Setting Up Your Client Application

The Bulk API uses HTTP GET and HTTP POST methods to send and receive CSV and XML content, so it's very 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.com.

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/v25.0/
-H "Authorization: Bearer
00D50000000IehZ\!AQcAQH0dMHZfz972Szmpkb58urFRkgeBGsxL_QJWwYMfAbUeeG7c1E6
LYUfiDUkWe6H34r1AAwOR8B8fLEz6n04NPGRrq0FM"
```

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

```
curl https://instance_name.salesforce.com/services/data/v25.0/
-H 'Authorization: Bearer sessionID'
```

Sending 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 the 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 57.



Note: Some examples in this document use a custom `Widget` object, which is described in [Getting Started with Database.com](#).

See Also:

[About URIs](#)

[Data Loader Guide](#)

Step 1: Logging In Using the SOAP SOAP API

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

To log in to Database.com using cURL:

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 Database.com user name and password.
3. Using a command-line window, execute the following cURL command:

```
curl https://login.database.com/services/Soap/u/25.0 -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. Database.com 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:

[Setting a Session Header](#)
[SOAP API Developer's Guide](#)

Step 2: Creating a Job

Before you can load any data, you first have to create a job. The job specifies the type of object, such as a custom Widget object, 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.

To create a job using cURL:

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/asynccapi/dataload">
  <operation>insert</operation>
  <object>Widget__c</object>
  <contentType>CSV</contentType>
</jobInfo>
```



Caution: 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/asynccapi/25.0/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.

Database.com 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/asynccapi/dataload">
  <id>750x0000000005LAAQ</id>
  <operation>insert</operation>
  <object>Widget__c</object>
  <createdById>005x00000000wPWdAAM</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>25.0</apiVersion>
</jobInfo>
```

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

See Also:

[Creating a New Job](#)

Step 3: Adding a Batch to the Job

After creating the job, you're now 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 with `jobId/batch` to the URI.

Format the data as either CSV or XML. For information about batch size limitations, see [Batch size and limits](#) on page 55.

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:

```
Name,Widget_Cost__c
Widget1,9.75
Widget2,3.45
```

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

```
curl https://instance.salesforce.com/services/asyncc/25.0/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.

Database.com 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/asynccapi/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>
</batchInfo>
```

Database.com 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:

[Preparing CSV Files](#)
[Adding a Batch to a Job](#)
[Bulk API Limits](#)

Step 4: Closing the Job

When you're finished submitting batches to Database.com, close the job. This informs Database.com that no more batches will be submitted for the job, which, in turn, allows the monitoring page in Database.com to return more meaningful statistics on the progress of the job.

To close a job using cURL:

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/asynccapi/dataload">
  <state>Closed</state>
</jobInfo>
```

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

```
curl https://instance.salesforce.com/services/asynccapi/25.0/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:

[Closing a Job](#)

Step 5: Checking Batch Status

You can check the status of an individual batch by running the following cURL command:

```
curl https://instance.salesforce.com/services/async/25.0/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.

Database.com 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/asynccapi/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>
</batchInfo>
```

If Database.com couldn't read the batch content or if the batch contained errors, such as invalid field names in the CSV header row, the batch state is `Failed`. When batch state is `Completed`, all records in the batch have been processed. However, individual records may 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/25.0/job/jobId/batch -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.

See Also:

[Getting Information for a Batch](#)

[Getting Information for All Batches in a Job](#)

[Interpreting Batch State](#)

Step 6: Retrieving 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/25.0/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.

Database.com returns a response with data such as the following:

```
"Id", "Success", "Created", "Error"  
"a01x0000004ouM4AAI", "true", "true", ""  
"a01x0000004ouM5AAI", "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:

[Getting Batch Results](#)

[Handling Failed Records in Batches](#)

Chapter 3

Planning Bulk Data Loads

In this chapter ...

- [General Guidelines for Data Loads](#)

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

This section gives you 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 they 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, if you had a custom object that had a relationship dependency on another custom object, when updating this custom object, the parent object would be locked during the transaction. If you load many batches of custom object records that all contain references to the same parent, they all try to lock the same parent 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 your batches by parent object ID so that all records referencing the same parent 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 [Getting Batch Results](#) on page 38. 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.com 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.com 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.com 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 [Monitoring a Batch](#) on page 33.

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 54.



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.

Chapter 4

Preparing Data Files

In this chapter ...

- [Finding Field Names](#)
- [Valid Date Format in Records](#)
- [Preparing CSV Files](#)
- [Preparing XML Files](#)

The Bulk API processes records in comma-separated values (*CSV*) files or *XML* files. This section tells you how to prepare your batches for processing.

Finding Field Names

Whether you're using CSV or XML data files, you need the names of the object fields that you want to update for the records in your data file. All the records in a data file must be for the same object. There are a few different ways to determine the field names for an object. You can:

- Use the `describeSObjects()` call in the *SOAP API Developer's Guide*.
- [Use Database.com Setup](#).

Use Database.com Setup

To find a field name for a custom object:

1. Click **Create > Objects**.
2. Click the `Label` for a custom object.
3. Click the `Field Label` for a field.

Use the `API Name` value as the field column header in a CSV file or the field name identifier in an XML file.

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](#)

Preparing 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 Database.com. 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.

See Also:

[Data Loader Guide](#)

Relationship Fields in a Header Row

You may have objects related to other objects in your Database.com instance. 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 `describeObjects()` call in the SOAP-based SOAP API to get the `relationshipName` property value for a field.

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.

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 Database.com SOQL and SOSL Reference Guide](#) at www.salesforce.com/us/developer/docs/dbcom_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, a custom Activity object could have a parent that is either a Person or a Calendar. In other words, the `Belongs To` field of an activity can contain the ID of either a person or a company. 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 `Belongs To` field is polymorphic and has a `relationshipName` of `Belongs_To__r`. It refers to a Person record and the indexed `External ID` field uniquely identifies the parent record.
2. The `Location` field is not polymorphic and has a `relationshipName` of `Location__r`. It refers to a custom Location object record and the indexed `External ID` field uniquely identifies the parent record.

```
Name,Person__c:Belongs_To__r.External_ID__c,Location__r.External_ID__c
Activity1,123445,99837556
```



Caution: 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 Database.com 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 Employee custom 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 records, 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__c,LastName__c,Title__c,ReportsTo__r.Email__c,Birthdate__c,Description__c
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](#)

[Data Loader Guide](#)

Preparing XML Files

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

All the records in an XML 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.

Note the following when processing XML 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, any fields that aren't defined in the XML file are ignored during the update.
- Files must be in UTF-8 format.

Relationship Fields in Records

You may have objects related to other objects in your Database.com instance. Some objects also have relationships to themselves. For example, you could have a custom object representing an employee that has a `Reports To` field that references another employee.

You can add a reference to a related object for a field in an XML 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>rwiliams@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.

Note the following when using relationships in XML records:

- You can use a child-to-parent relationship, but you can't use a parent-to-child relationship.
- You can use a child-to-parent relationship, but you can't extend it to use a child-to-parent-grandparent relationship.

Relationship Fields for Custom Objects

Custom objects use custom fields to track relationships between objects. Use the relationship name, which ends in `__r` (underscore-underscore-r), to represent a relationship between two custom objects. You can add a reference to a related object by using an indexed field. A custom field is indexed if its `External ID` field is selected.

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 *Database.com SOQL and SOSL Reference Guide* at www.salesforce.com/us/developer/docs/dbcom_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, a custom Activity object could have a parent that is either a Person or a Calendar. In other words, the `Belongs To` field of an activity can contain the ID of either a person or a company. 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>rwiliams@salesforcesample.com</IndexedFieldName>
  </sObject>
</RelationshipName>
```

The following sample includes two reference fields:

1. The `Belongs To` field is polymorphic and has a relationshipName of `Belongs_To__r`. It refers to a Person record and the indexed `External_ID` field uniquely identifies the parent record.
2. The `Location` field is not polymorphic and has a relationshipName of `Location__r`. It refers to a custom Location object record and the indexed `External_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>
    <Name>Activity1</Name>
    <Belongs_To__r>
      <sObject>
        <type>Person__c</type>
        <External_ID__c>123445</External_ID__c>
      </sObject>
    </Belongs_To__r>
    <Location__r>
      <sObject>
        <External_ID__c>99837556</External_ID__c>
      </sObject>
    </Location__r>
  </sObject>
</sObjects>
```

```
</sObject>
</sObjects>
```



Caution: 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 Records

A batch request in an XML file contains records for one object type. Each batch 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/asynccapi/dataload">
  <sObject>
    <field_name>field_value</field_name>
    ...
  </sObject>
  <sObject>
    <field_name>field_value</field_name>
    ...
  </sObject>
</sObjects>
```



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, a custom Activity object could have a parent that is either a Person or a Calendar. In other words, the `Belongs To` field of an activity can contain the ID of either a person or a company.

Note the following when generating records in XML files:

- Field values aren't trimmed. White space characters at the beginning or end of a field value are included in the saved value. For example, `John Smith` is saved with the leading and trailing space.
- Fields that aren't defined in the XML for a record are ignored when you update records. To set a field value to null, set the `xsi:nil="true"` value for the field to true. For example, `<description xsi:nil="true"/>` sets the description field to 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 a custom Employee 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 records, 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/asynccapi/dataload">
  <sObject>
    <FirstName__c>Tom</FirstName__c>
    <LastName__c>Jones</LastName__c>
    <Title__c>Senior Director</Title__c>
    <ReportsTo__r>
```

```
<sObject>
  <Email__c>buyer@salesforcesample.com</Email__c>
</sObject>
</ReportsTo__r>
<Birthdate__c>1940-06-07Z</Birthdate__c>
<Description__c>Branding guru</Description__c>
</sObject>
<sObject>
  <FirstName__c>Ian</FirstName__c>
  <LastName__c>Dury</LastName__c>
  <Title__c>Chief Imagineer</Title__c>
  <ReportsTo__r>
    <sObject>
      <Email__c>cto@salesforcesample.com</Email__c>
    </sObject>
  </ReportsTo__r>
  <Description__c>Expert in fuzzy logic design</Description__c>
</sObject>
</sObjects>
```

See Also:[Sample CSV File](#)

Chapter 5

Request Basics

In this chapter ...

- [About URIs](#)
- [Setting a Session Header](#)

This section describes some basics about the Bulk API, 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 the 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 25.0 of the Bulk API, your base URI would be

`https://na5.salesforce.com/services/async/25.0.`

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

See Also:

[Working with Jobs](#)

[Working with Batches](#)

Setting a Session Header

All HTTP requests must contain a valid API session ID obtained with the SOAP 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/25.0/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>Widget__c</object>
</jobInfo>
```

See Also:

[Quick Start](#)

[Sample Client Application Using Java](#)

Chapter 6

Working with Jobs

In this chapter ...

- [Creating a New Job](#)
- [Monitoring a Job](#)
- [Closing a Job](#)
- [Getting Job Details](#)
- [Aborting a Job](#)
- [Job and Batch Lifespan](#)

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.

Creating a New Job

Create a new 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 request body

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

In this sample, the `contentType` field indicates that the batches associated with the job are in CSV format. For alternative options, such as XML, see [JobInfo](#) on page 47.



Caution: The operation value must be all lower case. For example, you get an error if you use `INSERT` instead of `insert`.

Example response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
  xmlns="http://www.force.com/2009/06/asynccapi/dataload">
  <id>750D00000000021IAA</id>
  <operation>insert</operation>
  <object>CustomObject__c</object>
  <createdById>005D0000001ALVFIA4</createdById>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
  <state>Open</state>
  <contentType>CSV</contentType>
</jobInfo>
```

See Also:

- [Getting Job Details](#)
- [Closing a Job](#)
- [Aborting a Job](#)
- [Adding a Batch to a Job](#)
- [Job and Batch Lifespan](#)
- [Bulk API Limits](#)
- [About URIs](#)
- [JobInfo](#)
- [Quick Start](#)

Monitoring a Job

You can monitor a Bulk API job in Database.com. 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, click **Monitoring > Bulk Data Load Jobs**.

For more information, see “Monitoring Bulk Data Load Jobs” in the Database.com online help.

See Also:

- [Creating a New Job](#)
- [Getting Job Details](#)
- [Closing a Job](#)
- [Aborting a Job](#)
- [Adding a Batch to a Job](#)
- [Job and Batch Lifespan](#)
- [Bulk API Limits](#)
- [Data Loader Guide](#)

Closing 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 request body

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

Example response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
  xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D00000000021IAA</id>
  <operation>insert</operation>
  <object>CustomObject__c</object>
  <createdById>005D00000001ALVFIA4</createdById>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
```

```
<state>Closed</state>
</jobInfo>
```

See Also:

[Creating a New Job](#)
[Monitoring a Job](#)
[Getting Job Details](#)
[Aborting a Job](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[About URIs](#)
[JobInfo](#)
[Quick Start](#)

Getting 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 response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
  xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D00000000021IAA</id>
  <operation>insert</operation>
  <object>CustomObject__c</object>
  <createdById>005D00000001ALVFIA4</createdById>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
```

```
<state>Closed</state>
</jobInfo>
```

See Also:

[Creating a New Job](#)
[Monitoring a Job](#)
[Closing a Job](#)
[Aborting a Job](#)
[Adding a Batch to a Job](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[About URIs](#)
[JobInfo](#)
[Quick Start](#)

Aborting 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. Changes to data may already have been committed and aren't rolled back.

URI

`https://instance_name-api.salesforce.com/services/async/APIversion/job/jobId`

Example request body

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

Example response body

```
<?xml version="1.0" encoding="UTF-8"?>
<jobInfo
  xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>750D00000000021IAA</id>
  <operation>insert</operation>
  <object>CustomObject__c</object>
  <createdById>005D00000001ALVFIA4</createdById>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:16:00.000Z</systemModstamp>
```

```
<state>Aborted</state>  
</jobInfo>
```

See Also:

[Getting Job Details](#)
[Creating a New Job](#)
[Monitoring a Job](#)
[Closing 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 54.

See Also:

[Creating a New Job](#)
[Monitoring a Job](#)
[Getting Job Details](#)
[Closing a Job](#)
[Aborting a Job](#)
[Adding a Batch to a Job](#)
[Bulk API Limits](#)
[About URIs](#)
[JobInfo](#)
[Quick Start](#)

Chapter 7

Working with Batches

In this chapter ...

- [Adding a Batch to a Job](#)
- [Monitoring a Batch](#)
- [Getting Information for a Batch](#)
- [Getting Information for All Batches in a Job](#)
- [Interpreting Batch State](#)
- [Getting a Batch Request](#)
- [Getting Batch Results](#)
- [Handling Failed Records in Batches](#)

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

A batch is created by submitting a CSV or XML representation of a set of records and any references to binary attachments in an HTTP POST request. Once 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.

Adding 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 request body

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <sObject>
    <Name>[Bulk API] CustomObject 0 (batch 0)</Name>
    <Description__c>Created from Bulk API on Tue Apr 14 11:15:59 PDT 2009</Description__c>
  </sObject>
  <sObject>
    <Name>[Bulk API] CustomObject 1 (batch 0)</Name>
    <Description__c>Created from Bulk API on Tue Apr 14 11:15:59 PDT 2009</Description__c>
  </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, see [JobInfo](#) on page 47.

Example response body

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
  xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <id>751D0000000004rIAA</id>
  <jobId>750D00000000021IAA</jobId>
  <state>Queued</state>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
```

```
<numberRecordsProcessed>0</numberRecordsProcessed>  
</batchInfo>
```

See Also:

[Getting Information for a Batch](#)
[Monitoring a Batch](#)
[Getting Information for All Batches in a Job](#)
[Interpreting Batch State](#)
[Getting a Batch Request](#)
[Getting Batch Results](#)
[Working with Jobs](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[About URIs](#)
[BatchInfo](#)
[Quick Start](#)

Monitoring a Batch

You can monitor a Bulk API batch in Database.com.

To track the status of bulk data load jobs and their associated batches, click **Monitoring > Bulk Data Load Jobs**. Click on 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 file, the links return the request or response in XML format. 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 Database.com online help.

See Also:

[Getting Information for a Batch](#)
[Adding a Batch to a Job](#)
[Getting Information for All Batches in a Job](#)
[Interpreting Batch State](#)
[Getting a Batch Request](#)
[Getting Batch Results](#)
[Handling Failed Records in Batches](#)
[Working with Jobs](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[About URIs](#)
[BatchInfo](#)
[Quick Start](#)

Getting 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 response body

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfo
  xmlns="http://www.force.com/2009/06/asynccapi/dataload">
  <id>751D00000000004rIAA</id>
  <jobId>750D000000000021IAA</jobId>
  <state>InProgress</state>
  <createdDate>2009-04-14T18:15:59.000Z</createdDate>
  <systemModstamp>2009-04-14T18:15:59.000Z</systemModstamp>
```

```
<numberRecordsProcessed>0</numberRecordsProcessed>
</batchInfo>
```

See Also:

[Adding a Batch to a Job](#)
[Monitoring a Batch](#)
[Getting Information for All Batches in a Job](#)
[Interpreting Batch State](#)
[Getting a Batch Request](#)
[Getting Batch Results](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[BatchInfo](#)
[About URIs](#)
[Working with Jobs](#)
[Quick Start](#)

Getting 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 response body

```
<?xml version="1.0" encoding="UTF-8"?>
<batchInfoList
  xmlns="http://www.force.com/2009/06/asynccapi/dataload">
  <batchInfo>
    <id>751D00000000004rIAA</id>
    <jobId>750D000000000021IAA</jobId>
    <state>InProgress</state>
    <createdDate>2009-04-14T18:15:59.000Z</createdDate>
    <systemModstamp>2009-04-14T18:16:09.000Z</systemModstamp>
    <numberRecordsProcessed>800</numberRecordsProcessed>
  </batchInfo>
  <batchInfo>
    <id>751D000000000004sIAA</id>
    <jobId>750D000000000021IAA</jobId>
    <state>InProgress</state>
    <createdDate>2009-04-14T18:16:00.000Z</createdDate>
    <systemModstamp>2009-04-14T18:16:09.000Z</systemModstamp>
    <numberRecordsProcessed>800</numberRecordsProcessed>
  </batchInfo>
</batchInfoList>
```

```
</batchInfo>  
</batchInfoList>
```

See Also:

[Adding a Batch to a Job](#)
[Monitoring a Batch](#)
[Getting Information for a Batch](#)
[Interpreting Batch State](#)
[Getting a Batch Request](#)
[Getting Batch Results](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[BatchInfo](#)
[About URIs](#)
[Working with Jobs](#)
[Quick Start](#)

Interpreting 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, this batch isn't processed and its state is set to `Not Processed`.

InProgress

The batch is currently being processed. If the job associated with this batch is aborted, this batch is still processed to completion.

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 being 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.

See Also:

[Adding a Batch to a Job](#)
[Monitoring a Batch](#)
[Getting Information for All Batches in a Job](#)
[Getting a Batch Request](#)
[Getting Batch Results](#)
[Handling Failed Records in Batches](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[BatchInfo](#)
[About URIs](#)
[Working with Jobs](#)
[Quick Start](#)

Getting a Batch Request

Get a batch request by sending a GET request to the following URI.

Alternatively, you can get a batch request in Database.com. To track the status of bulk data load jobs and their associated batches, click **Monitoring > Bulk Data Load Jobs**. Click on 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 file, the links return the request or response in XML format.



Note: Available in API version 19.0 and later.

URI

`https://instance_name-api.salesforce.com/services/async/APIversion/job/jobid/batch/batchId/request`

Example request body

No request body is allowed.

Example response body

```
<?xml version="1.0" encoding="UTF-8"?>
<sObjects xmlns="http://www.force.com/2009/06/asyncapi/dataload">
  <sObject>
    <Name>[Bulk API] CustomObject 0 (batch 0)</Name>
    <Description__c>Created from Bulk API on Tue Apr 14 11:15:59 PDT 2009</Description__c>
  </sObject>
  <sObject>
    <Name>[Bulk API] CustomObject 1 (batch 0)</Name>
    <Description__c>Created from Bulk API on Tue Apr 14 11:15:59 PDT 2009</Description__c>
  </sObject>
</sObjects>
```

```
</sObject>
</sObjects>
```

See Also:

[Getting Information for a Batch](#)
[Monitoring a Batch](#)
[Getting Information for All Batches in a Job](#)
[Interpreting Batch State](#)
[Getting Batch Results](#)
[Working with Jobs](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[About URIs](#)
[BatchInfo](#)
[Quick Start](#)

Getting 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 file, the response is in XML format.

Alternatively, you can monitor a Bulk API batch in Database.com. To track the status of bulk data load jobs and their associated batches, click **Monitoring > Bulk Data Load Jobs**. Click on 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 file, the links return the request or response in XML format. 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>a01D000000ISUr3IAH</id><success>true</success><created>true</created>
  </result>
  <result>
    <id>a01D000000ISUr4IAH</id><success>true</success><created>true</created>
  </result>
</results>
```


For a CSV Batch:

```
"Id","Success","Created","Error"
"a01D000000ISUr3IAH","true","true",""
"a01D000000ISUr4IAH","true","true",""
","false","false","REQUIRED_FIELD_MISSING:Required fields are missing:
[LastName__c]:LastName__c --"
```



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 [Handling Failed Records in Batches](#) on page 39.

See Also:

[Adding a Batch to a Job](#)
[Monitoring a Batch](#)
[Getting a Batch Request](#)
[Getting Information for a Batch](#)
[Getting Information for All Batches in a Job](#)
[Interpreting Batch State](#)
[Job and Batch Lifespan](#)
[Bulk API Limits](#)
[BatchInfo](#)
[About URIs](#)
[Working with Jobs](#)
[Quick Start](#)

Handling 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 [Getting Information for a Batch](#) on page 34 and [Interpreting Batch State](#) on page 36.
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"
"a01D000000ISUr3IAH","true","true",""
"a01D000000ISUr4IAH","true","true",""
```

```
"", "false", "false", "REQUIRED_FIELD_MISSING:Required fields are missing:  
[LastName__c]:LastName__c --"
```

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 Database.com.

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:

[Adding a Batch to a Job](#)

[Errors](#)

[Bulk API Limits](#)

Chapter 8

Bulk Query

In this chapter ...

- [Bulk Query Details](#)

This section describes Bulk API queries, including information on errors and processing limits.

Bulk Query Details

Using bulk query reduces the number of API requests and is more efficient for large data sets. A maximum of 10 gigabytes of data can be retrieved by the query, divided into 10 files. The data formats supported are XML and CSV. This table summarizes the prominent features of the bulk query:

Feature	Functionality
Retrieved file size	1 gigabyte
Number of retrieved files	10 files ¹
Number of attempts to query	10 attempts at 10 minutes each ²
Length of time results are kept	7 days

URI

`https://instance_name-api.salesforce.com/services/async/APIversion/job/jobid/batch`

Bulk Query Request

```
POST baseURI/job/750x00000000014/batch
X-SFDC-Session: 4fla00D30000000K7zB!ARUAQDqAHcM...
Content-Encoding: gzip

[octet stream of gzipped SOQL query]
```



Note:

Bulk API query doesn't support the following SOQL:

- COUNT
- ROLLUP
- SUM
- GROUP BY CUBE
- OFFSET
- Nested SOQL queries, for example:

```
SELECT Activity__c.Name, (SELECT Employee__c.FirstName__c,
Employee__c.LastName
FROM Activity__c.Owners__c) FROM Activity__c
```

- Relationship fields

SOAP Headers, Requests and Responses

The following are example Bulk Query requests and responses.

¹ If the query needs to return more than 10 files, the query should be filtered to return less data. Bulk batch sizes are not used for bulk queries.

² If more than 10 attempts are made for the query, an error message of "Tried more than ten times" is returned.

Create Bulk Query Batch HTTP Request

```
POST baseURI/job/750x00000000014/batch
X-SFDC-Session: 4f1a00D30000000K7zB!ARUAQDqAHcM...

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/asynccapi/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>
</batchInfo>
```

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/asynccapi/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"
```

Java Example Using WSC

In the following example, login in to your organization using the Partner API, then instantiate 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:

[Walk Through the Bulk Query Sample](#)

Chapter 9

Reference

In this chapter ...

- [Schema](#)
- [JobInfo](#)
- [BatchInfo](#)
- [HTTP Status Codes](#)
- [Errors](#)
- [Bulk API Limits](#)

This section describes 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 25.0 of the Bulk API, the URI is:

```
https://na5.salesforce.com/services/async/25.0/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 Database.com for processing. When a job is created, Database.com 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
<code>apiActiveProcessingTime</code>	<code>long</code>	Do not specify for new job.	<p>The number of milliseconds taken to actively process the job and includes <code>apexProcessingTime</code>, 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.</p> <p>This field is available in API version 19.0 and later.</p>
<code>concurrencyMode</code>	<code>ConcurrencyModeEnum</code>		<p>The concurrency mode for the job. The valid values are:</p> <ul style="list-style-type: none"> <code>Parallel</code>: Process batches in parallel mode. This is the default value. <code>Serial</code>: 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 <code>serial</code> 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.
<code>contentType</code>	<code>ContentType</code>		<p>The content type for the job. The valid values are:</p> <ul style="list-style-type: none"> <code>CSV</code>—data in CSV format <code>XML</code>—data in XML format (default option) <code>ZIP_CSV</code>—data in CSV format in a zip file containing binary attachments <code>ZIP_XML</code>—data in XML format in a zip file containing binary attachments
<code>createdById</code>	<code>string</code>	System field	The ID of the user who created this job. All batches must be created by this same user.
<code>createdDate</code>	<code>dateTime</code>	System field	The date and time in the UTC time zone when the job was created.
<code>externalIdFieldName</code>	<code>string</code>	Required with <code>upsert</code>	The name of the external ID field for an <code>upsert()</code> .
<code>id</code>	<code>string</code>	Do not specify for new job.	<p>Unique, 18-character ID for this job.</p> <p>All GET operations return this value in results.</p>
<code>numberBatchesCompleted</code>	<code>int</code>	Do not specify for new job.	The number of batches that have been completed for this job.
<code>numberBatchesQueued</code>	<code>int</code>	Do not specify for new job.	The number of batches queued for this job.

Name	Type	Request	Description
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.	<p>The number of total batches currently in the job. This value increases as more batches are added to the job. When the jobstate is Closed or Failed, this number represents the final total.</p> <p>The job is complete when numberBatchesTotal equals the sum of numberBatchesCompleted and numberBatchesFailed.</p>
numberRecordsFailed	int	Do not specify for new job.	<p>The number of records that were not processed successfully in this job.</p> <p>This field is available in API version 19.0 and later.</p>
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 Database.com 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	<p>The processing operation for all the batches in the job. The valid values are:</p> <ul style="list-style-type: none"> • delete • insert • query • upsert • update • hardDelete <p> Caution: The operation value must be all lower case. For example, you get an error if you use INSERT instead of insert.</p> <p>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.</p> <p> Caution: 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</p>

Name	Type	Request	Description
			Hard Delete,” is disabled by default and must be enabled by an administrator. A user license is required for hard delete.
state	JobStateEnum	Required if creating, closing, or aborting a job.	<p>The current state of processing for the job:</p> <ul style="list-style-type: none"> • 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.
totalProcessingTime	long	Do not specify for new job.	<p>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.</p> <p>This field is available in API version 19.0 and later.</p>

See Also:[Working with Jobs](#)[Quick Start](#)[SOAP API Developer's Guide](#)

BatchInfo

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

BatchInfo

Name	Type	Request	Description
apexProcessingTime	long	System field	<p>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.</p> <p>This field is available in API version 19.0 and later.</p>
apiActiveProcessingTime	long	System field	<p>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.</p> <p>This field is available in API version 19.0 and later.</p>
createdDate	dateTime	System field	The date and time in the UTC time zone when the batch was created. This is not the time processing began, but the time the batch was added to the job.
id	string	Required	The ID of the batch. May be globally unique, but does not have to be.
jobId	string	Required	The unique, 18-character ID for the job associated with this batch.
numberRecordsFailed	int	System field	<p>The number of records that were not processed successfully in this batch.</p> <p>This field is available in API version 19.0 and later.</p>
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	<p>The current state of processing for the batch:</p> <ul style="list-style-type: none"> • Queued: Processing of the batch has not started yet. If the job associated with this batch is aborted, this batch isn't processed and its state is set to <code>Not Processed</code>. • InProgress: The batch is currently being processed. If the job associated with this batch is aborted, this batch is still processed to completion. • 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 being compressed with an unsupported format, or an internal server error. The <code>stateMessage</code> element may contain more details about any failures. Even if the batch failed, some records could

Name	Type	Request	Description
			<p>have been completed successfully. The <code>numberRecordsProcessed</code> field tells you how many records were processed. The <code>numberRecordsFailed</code> field contains the number of records that were not processed successfully.</p> <ul style="list-style-type: none"> • <code>Not Processed</code>: The batch won't be processed. This state is assigned when a job is aborted while the batch is queued.
<code>stateMessage</code>	string	System field	Contains details about the state. For example, if the <code>state</code> value is <code>Failed</code> , 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.
<code>systemModstamp</code>	dateTime	System field	The date and time in the UTC time zone that processing ended. This is only valid when the state is <code>Completed</code> .
<code>totalProcessingTime</code>	long	System field	<p>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.</p> <p>This field is available in API version 19.0 and later.</p>

HTTP BatchInfoList

Name	Type	Description
<code>batchInfo</code>	BatchInfo	One BatchInfo resource for each batch in the associated job. For the structure of BatchInfo, see BatchInfo on page 51.

See Also:

[Working with Batches](#)

[Interpreting Batch State](#)

[Quick Start](#)

[SOAP API Developer's Guide](#)

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

The Bulk API is not enabled for this organization.

InvalidBatch

The batch ID specified in a batch update or query is invalid.

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 Database.com takes too long to process a batch. For more information on timeout limits, see [Batch processing time](#) on page 55. 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 the SOAP API. For more information on these codes, see “ExceptionCode” in the [SOAP API Developer's Guide](#).

See Also:

[HTTP Status Codes](#)

[Handling Failed Records in Batches](#)

Bulk API Limits

Please 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 or XML 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 2,000 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 or XML file that can be 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.

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 [Getting Batch Results](#) on page 38. If you get a timeout error when processing a batch, split your batch into smaller batches, and try again.

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 Database.com.

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	10 files ³
Number of attempts to query	10 attempts at 10 minutes each ⁴
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, for example:

```
SELECT Activity__c.Name, (SELECT Employee__c.FirstName__c, Employee__c.LastName
FROM Activity__c.Owners__c) FROM Activity__c
```

- Relationship fields

³ If the query needs to return more than 10 files, the query should be filtered to return less data. Bulk batch sizes are not used for bulk queries.

⁴ If more than 10 attempts are made for the query, an error message of "Tried more than ten times" is returned.

Appendix A

Sample Client Application Using Java

Use the code sample in this section 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 copy 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.
- Some examples in this document use a custom `Widget` object, which is described in [Getting Started with Database.com](#).

Set Up Your Client Application

The Bulk API uses HTTP GET and HTTP POST methods to send and receive XML content, so it's very simple to build clients in the language of your choice. This section uses a Java sample and the Salesforce Web Service Connector (WSC) toolkit provided by [salesforce.com](#) 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:

<http://code.google.com/p/sfdc-wsc/>

To download the Salesforce WSC toolkit:

1. Browse to <http://code.google.com/p/sfdc-wsc/>
2. Click the **Downloads** tab.
3. Click the `wsc-19.jar` link 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 Database.com 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 “Setting Login Restrictions” in the Database.com online help.

2. Navigate to **Develop > 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

Once you have set up your client, you can begin building 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 Widget custom object records
    example.runSample("Widget__c", "myUser@myOrg.com", "myPassword", "mySampleData.csv");
}

/**
 * Creates a Bulk API job and uploads batches for a CSV file.
 */
public void runSample(String objectType, String userName,
    String password, String sampleFileName)
    throws AsyncApiException, ConnectionException, IOException {
    BulkConnection connection = getBulkConnection(userName, password);
```

```

        JobInfo job = createJob(subjectType, 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.database.com/services/Soap/u/25.0");
    // 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 = "25.0";
    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 life span.

Create a New Job

After creating the connection, create a new job. Data is always processed in the context of a job. The job specifies the details about the data being processed: what operation is being executed (insert, update, upsert, or delete) and the object type. The following code creates a new insert job on the Widget custom object.

```

/**
 * Create a new job using the Bulk API.
 *
 * @param subjectType
 *         The object type being loaded, such as "Widget__c"
 * @param connection

```

```

    *           BulkConnection used to create the new job.
    * @return The JobInfo for the new job.
    * @throws AsyncApiException
    */
private JobInfo createJob(String objectType, BulkConnection connection)
    throws AsyncApiException {
    JobInfo job = new JobInfo();
    job.setObject(objectType);
    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. Once 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 54.

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 Database.com.

```

/**
 * 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();
    }
}

```

Once the server receives a batch it's immediately queued for processing. Any errors in formatting aren't reported when sending the batch. These errors are reported in the result data when the batch is processed.

Close the Job

After all batches have been added to a job, close the job.

```
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 Widget custom object records
        example.runSample("Widget__c", "myUser@myOrg.com", "myPassword", "mySampleData.csv");
    }

    /**
     * Creates a Bulk API job and uploads batches for a CSV file.
     */
    public void runSample(String objectType, String userName,
        String password, String sampleFileName)
        throws AsyncApiException, ConnectionException, IOException {
        BulkConnection connection = getBulkConnection(userName, password);
        JobInfo job = createJob(objectType, 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 objectType
 *         The object type being loaded, such as "Widget__c"
 * @param connection
 *         BulkConnection used to create the new job.
 * @return The JobInfo for the new job.
 * @throws AsyncApiException
 */
private JobInfo createJob(String objectType, BulkConnection connection)
    throws AsyncApiException {
    JobInfo job = new JobInfo();
    job.setObject(objectType);
    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.database.com/services/Soap/u/25.0");
    // 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 = "25.0";
    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

Sample Bulk Query Using cURL

The code sample in this section uses cURL to query a number of account records using the REST-based Bulk API.

This section consists of step-by-step instructions.



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.

Walk Through the Bulk Query Sample

Create a Job

The following example shows how to use bulk query from the command line using cURL.

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/asynccapi/dataload">
  <operation>query</operation>
  <object>Widget__c</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/asynccapi/25.0/job
```

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

Database.com 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/asynccapi/dataload">
<id>750x000000009tvAAA</id>
<operation>query</operation>
<object>Widget__c</object>
<createdById>005x0000001WR01AAG</createdById>
<createdDate>2011-03-10T00:53:19.000Z</createdDate>
<systemModstamp>2011-03-10T00:53:19.000Z</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>21.0</apiVersion>
<numberRecordsFailed>0</numberRecordsFailed>
<totalProcessingTime>0</totalProcessingTime>
<apiActiveProcessingTime>0</apiActiveProcessingTime>
<apexProcessingTime>0</apexProcessingTime>
</jobInfo>

```

Add the Job to a Batch

1. Create a file called `query.txt` to contain the SOQL query statement.

```
SELECT Name, Widget__Cost__c FROM Widget__c LIMIT 10
```

2. Using a command-line window, execute the following cURL command to add the job to a batch:

```
curl -d @query.txt -H "X-SFDC-Session: sessionId" -H "Content-Type: text/csv; charset=UTF-8" https://instance.salesforce.com/services/async/25.0/job/jobId/batch
```

jobId is the job ID returned in the response to the job creation.

Database.com 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/asynccapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Queued</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T00:59:47.000Z</systemModstamp>
  <numberRecordsProcessed>0</numberRecordsProcessed>
  <numberRecordsFailed>0</numberRecordsFailed>
  <totalProcessingTime>0</totalProcessingTime>
  <apiActiveProcessingTime>0</apiActiveProcessingTime>
  <apexProcessingTime>0</apexProcessingTime>
</batchInfo>

```

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/25.0/job/jobId
```


Database.com 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/asynccapi/dataload">
  <id>750x000000009tvAAA</id>
  <operation>query</operation>
  <object>Widget__c</object>
  <createdById>005x0000001WR01AAG</createdById>
  <createdDate>2011-03-10T00:53:19.000Z</createdDate>
  <systemModstamp>2011-03-10T00:53:19.000Z</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>21.0</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/25.0/job/jobId/batch/batchId
```

batchId is the batch ID in the response to the batch creation.

Database.com 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/asynccapi/dataload">
  <id>751x000000009vwAAA</id>
  <jobId>750x000000009tvAAA</jobId>
  <state>Completed</state>
  <createdDate>2011-03-10T00:59:47.000Z</createdDate>
  <systemModstamp>2011-03-10T01:00:19.000Z</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/25.0/job/jobId/batch/batchId/result
```

Database.com 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/25.0/job/jobId/batch/batchId/result/resultId
```

resultId is the result ID in the response to the batch result list request.

Database.com returns an XML response with data such as the following:

```
"Name", "Widget Cost__c"
"Widget 1", 9.45
"Widget 2", 6.75
"Widget 3", 3.75
```

[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 Database.com 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. Apex code can be initiated by Web service requests and from triggers on objects.

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 or XML 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 Database.com. See also SOAP API.

C

Client App

An app that runs outside the Database.com 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. See also Composite App and Native App.

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 Database.com 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 Database.com organization.

Database

An organized collection of information. The underlying architecture of Database.com 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.

Database.com 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 Force

The Developer Force website at developer.force.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**Entity Relationship Diagram (ERD)**

A data modeling tool that helps you organize your data into entities (or objects, as they are called in the Force.com platform) and define the relationships between them. ERD diagrams for key Database.com objects are published in the *SOAP API Developer's Guide*.

F**Field**

A part of an object that holds a specific piece of information, such as a text or currency value.

Force.com

The salesforce.com 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 Database.com 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 twelve month structure used throughout much of the world.

H

No Glossary items for this entry.

I**ID**

See 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. Database.com runs on multiple instances, but data for any single organization is always consolidated on a single instance.

Integration User

A Database.com 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](#).

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, 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 Database.com. 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**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, invoice statements have a master-detail relationship with line items. This type of relationship affects record deletion and security.

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 Database.com 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 invoice statement may have one or more line items.

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 action that sends the information you specify to an endpoint you designate, such as an external service. An outbound message sends the data in the specified fields in the form of a SOAP message to the endpoint. Outbound messaging is configured in the Database.com 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 is assigned.

P

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 Database.com 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 Database.com 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.

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 Database.com organization that has live users accessing data.

Q**Query String Parameter**

A name-value pair that's included in a URL, typically after a '?' character.

R**Record**

A single instance of a Database.com object.

Record Name

A standard field on all Database.com objects. 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 Console tab'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 SOA (Service-Oriented Architecture)

A powerful capability of Force.com that allows you to make calls to external Web services from within Apex.

Session ID

An authentication token that is returned when a user successfully logs in to Database.com. The Session ID prevents a user from having to log in again every time he or she wants to perform another action in Database.com. Different from a record ID or Database.com ID, which are terms for the unique ID of a Database.com 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 Database.com 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

An administration area where you can customize and define Force.com applications. Access Setup through the **Your Name > Setup** link at the top of Database.com pages.

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.

T**Test Database**

A nearly identical copy of a Database.com production organization. You can create a test database for a variety of purposes, such as testing and training, without compromising the data and applications in your production environment.

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 Database.com 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 Database.com in their language.

U**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 Database.com 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 Database.com 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 Database.com Enterprise WSDL or Partner WSDL to communicate with Database.com 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

A

Abort job 29
Add batch 8, 32
Attachments 15

B

Batch lifespan 30
Batches
 adding to a job 8, 32
 checking status 10
 creating 32
 getting a request 37
 getting information 34
 getting information for job 35
 getting results 38
 handling failed rows 39
 interpreting state 36
 monitoring 33
 retrieving results 10
Bulk Query 41–42

C

Checking batch status 10
Client application 57, 69
Close job 9, 27
Code sample
 setting up organization 5
 setting up your client 57
 walk through sample code 58, 69
Create batch 8, 32
Create job 7, 26
CSV
 external ID 17
 null values 18
 polymorphic field 17
 record rows 18
 relationship field 17
 sample file 18
CSV data files 16
cURL 6
cURL sample 69

D

Data files
 CSV 16
 date format 16
 finding field names 16
 introduction 15
 XML 19
Date format 16

E

Error codes 53
Errors
 batch results 39
External ID 17

F

Finding field names 16

G

Get job details 28
Get job status and results 28
Getting started sample 5, 57, 69
Guidelines 13

H

HTTP requests 5–6
HTTP status codes 52

I

Introduction 3

J

Java client application 57
Job lifespan 30
Jobs
 aborting 29
 closing 9, 27
 creating 7, 26
 getting details 28
 monitoring 27

L

Large data volumes 3
Limits 54
Lock contention 13
Logging in 24
Login 6
Longevity of jobs and batches 30

M

Monitor batch 33
Monitor job 27

O

Operations
 on jobs 25

P

Planning 12
Polymorphic field 17, 19

Q

Quick start
 cURL 5

Quick start (*continued*)

- setting up organization 5
- setting up your client 5

R

Reference

- batches (BatchInfo) 50
- error codes 53
- HTTP status codes 52
- jobs (JobInfo) 47

Request

- getting for batch 37

Resource 24

Results

- getting for batch 38
- handling failed rows 39

Retrieving batch results 10

S

Sample code 57, 69

Sample for quick start 5

Sample job operations 25

Schema 47

Session header 24

SOAP 6

State

- interpreting for batch 36

Status

- get for job 28
- getting for batch 34
- getting for batches 35
- interpreting for batch 36

Status codes 52

U

URI 24

W

WSC 57

WSDL 6

X

XML

- null values 21
- polymorphic field 19
- records 21
- relationship field 19
- sample file 21

XML data files 19

XSD 47