Version 23.0: Winter '12

Database.com REST API Developer's Guide

Last updated: November 19, 2011

Table of Contents

Getting Started with the Database.com REST API....3

Chapter 1: Introducing the Force.com REST API....3

Understanding Force.com REST Resources....4

Using Compression....5

Quick Start....6

Chapter 2: Step 1: Obtain an Organization....6

Chapter 3: Step 2: Create Objects and Fields....7

Create Widget Object....7

Create Model Object....8

Relate the Objects....8

Chapter 4: Step 3: Create a Remote Application....9

Chapter 5: Step 4: Walk Through the Sample Code....10

Java Sample Code....10

Using REST Resources....16

Chapter 6: Working with Objects and Records....16

List Available REST API Versions....17

List Available REST Resources....17

Get a List of Objects....18

Retrieve Metadata for an Object....18

Get Field and Other Metadata for an Object....19

Get Field Values from Records....20

Execute a SOQL Query....21

Search for a String....22

Create a Record....23

Update a Record....24

Insert or Update (Upsert) a Record....25

Delete a Record....26

Chapter 7: Reference....28

Versions....29

Resources by Version....29

Describe Global....30

SObject Basic Information....31

SObject Describe....33

SObject Rows....35

SObject Rows by External ID....37

Query....40
Search....42
Error Response....43

Index....45

GETTING STARTED WITH THE DATABASE.COM REST API

Chapter 1

Introducing the Force.com REST API

In this chapter ...

- Understanding Force.com REST Resources
- Using Compression

The Force.com REST API provides you with a powerful, convenient, and simple Web services API for interacting with Force.com. Its advantages include ease of integration and development, and it is an excellent choice of technology for use with mobile applications and Web 2.0 projects. However, if you have large numbers of records to process, you may wish to use the Bulk API, which is based on REST principles and optimized for large sets of data.

The REST API uses the same underlying data model and standard objects as those in the SOAP-based Web services API. See the Web Services API Developer's Guide for details. The REST API also has the same limits imposed on it as the Web Services API. See the Limits section in the Web Services API Developer's Guide.

To use this document, you should have a basic familiarity with software development, Web services, and the Database.com user interface.

Use this introduction to understand:

- The key characteristics and architecture of the Force.com REST API. This will help you understand how your applications can best use the Force.com REST resources.
- How to set up your development environment so you can begin working with the REST API immediately.
- How to use the REST API by following a quick start that leads you step by step through a typical use case.

Understanding Force.com REST Resources

Each resource in the Force.com REST API is a named URI that is used with an HTTP method (HEAD, GET, POST, PATCH, or DELETE). You use a resource to interact with your Database.com organization. For example, you can:

- Retrieve summary information about the API versions available to you.
- Obtain detailed information about a custom object.
- Perform a query or search.
- Update or delete records.

The Force.com REST API is based on the usage of resources, their URIs, and the links between them. The resources are accessed using a standard set of HTTP methods.

Suppose you would like to retrieve information about the Database.com version. To do this, submit a request for the Versions resource (this example uses cURL on the *na1* instance):

```
curl https://na1.salesforce.com/services/data/
```

The output from this request is as follows:

```
[
    {
        "version":"20.0",
        "url":"/services/data/v20.0",
        "label":"Winter '11"
    }
    ...
]
```

Note: Database.com runs on multiple server instances. The examples in this guide use the *na1* instance. The instance your organization uses may be different.

Important characteristics of the Force.com REST API resources and architecture:

Stateless

Each request from client to server must contain all the information necessary to understand the request, and not use any stored context on the server. However, the representations of the resources are interconnected using URLs, which allow the client to progress between states.

Caching behavior

Responses are labeled as cacheable or non-cacheable.

Uniform interface

All resources are accessed with a generic interface over HTTP.

Named resources

All resources are named using a base URI that follows your Force.com URI.

Layered components

The Force.com REST API architecture allows for the existence of such intermediaries as proxy servers and gateways to exist between the client and the resources.

Authentication

The Force.com REST API supports OAuth 2.0.

Support for JSON and XML

JSON is the default. You can use the `HTTP ACCEPT` header to select either JSON or XML, or append `.json` or `.xml` to the URI (for example, `/Widget__c/a01D000000INjVe.json`).

The JavaScript Object Notation (JSON) format is supported with UTF-8. Date-time information is in ISO8601 format.

XML serialization is similar to the SOAP-based Web services API. XML requests are supported in UTF-8 and UTF-16, and XML responses are provided in UTF-8.

Using Compression

The REST API allows the use of compression on the request and the response, using the standards defined by the HTTP 1.1 specification. Compression is automatically supported by some clients, and can be manually added to others. Visit Developer Force for more information on particular clients.

Tip: For better performance, we suggest that clients accept and support compression as defined by the HTTP 1.1 specification.

To use compression, include the HTTP header `Accept-Encoding: gzip` or `Accept-Encoding: deflate` in a request. The REST API compresses the response if the client properly specifies this header. The response includes the header `Content-Encoding: gzip` or `Accept-Encoding: deflate`. You can also compress any request by including a `Content-Encoding: gzip` or `Content-Encoding: deflate` header.

Response Compression

The REST API can optionally compress responses. Responses are compressed only if the client sends an `Accept-Encoding` header. The REST API is not required to compress the response even if you have specified `Accept-Encoding`, but it normally does. If the REST API compresses the response, it also specifies a `Content-Encoding` header.

Request Compression

Clients can also compress requests. The REST API decompresses any requests before processing. The client must send a `Content-Encoding` HTTP header in the request with the name of the appropriate compression algorithm. For more information, see:

- Content-Encoding at: `www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.11`
- Accept-Encoding at: `www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.3`
- Content Codings at: `www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.5`

QUICK START

Chapter 2

Step 1: Obtain an Organization

If you don't already have an account, go to www.database.com and follow the instructions for joining.

If you already have an organization, verify that you have the "API Enabled" permission. This permission is enabled by default, but may have been changed by an administrator. For more information, see the Salesforce online help.

Chapter 3

Step 2: Create Objects and Fields

In this step you'll create two objects, widget and model, each with a custom field. Then you'll relate the objects to each other with a one-to-many-relationship.

Create Widget Object

To create the widget object with a widget cost field:

1. Click **Create** > **Objects**.
2. Click **New Custom Object**.
3. Enter the information for the widget object:
   - Label: `Widget`
   - Plural label: `Widgets`
   - Object name: `Widget`
   - Record name: `Widget Name`
   - Data type: `Text`
4. Leave all other settings as they are and click **Save**.
5. In the Custom Fields & Relationships related list, click **New**.
6. For Data Type, select `Currency` and click **Next**.
7. Enter the custom field details.
   - Field Label: `Widget Cost`
   - Length: `10`
   - Decimal places: `2`
   - Field Name: `Widget_Cost`
8. Leave the remaining settings as they are and click **Next**.
9. Click **Save** to accept the default field-level security settings.

See Also:

Step 2: Create Objects and Fields
Create Model Object

Create Model Object

To create the model object with a model number field:

1. Click **Create** > **Objects**.
2. Click **New Custom Object**.
3. Enter the information for the model object:
   - Label: `Model`
   - Plural label: `Models`
   - Object name: `Model`
   - Record name: `Model Name`
   - Data type: `Text`
4. Leave all other settings as they are and click **Save**.
5. In the Custom Fields & Relationships related list, click **New**.
6. For Data Type, select `Text` and click **Next**.
7. Enter the custom field details.
   - Field Label: `Model Number`
   - Length: `10`
   - Field Name: `Model_Number`
8. Leave the remaining settings as they are and click **Next**.
9. Click **Save** to accept the default field-level security settings.

See Also:

Step 2: Create Objects and Fields
Create Widget Object
Relate the Objects

Relate the Objects

1. If you aren't already in the Model detail page, click **Create** > **Objects**, then select the Model object.
2. In the Custom Fields & Relationships related list, click **New**.
3. In the New Custom Field page, select `Master-Detail Relationship` and click **Next**.
4. In the Related To field, select the Widget object and click **Next**.
5. Accept the defaults on the remaining screens by clicking **Next** and then **Save**.

See Also:

Step 2: Create Objects and Fields
Create Model Object

Chapter 4

Step 3: Create a Remote Application

To create a remote access application for your organization:

1. Log in to your organization. Logins are checked to ensure they are from a known IP address.
2. Click **Develop** > **Remote Access** to display the Remote Access page.
3. Click **New**.
4. Enter the information for the remote access application:
   - Application: `MyRemoteAccessApplication`
   - Callback URL: `https://no_redirect_uri`
   - Contact Email: ***`your_email@domain.ext`***
5. Click **Save**.

Chapter 5

Step 4: Walk Through the Sample Code

Once you've created your remote application, you can begin building client applications that use the REST API. Use the following samples to create a basic client application. Comments embedded in the sample explain each section of code.

Java Sample Code

This section walks through a sample Java client application that uses the REST API. The purpose of this sample application is to show the required steps for logging into the login server and to demonstrate the invocation and subsequent handling of several REST API calls. This sample application performs the following main tasks:

1. Prompts the user for
   - API version
   - login URL
   - username
   - password
   - OAuth 2.0 consumer key
   - OAuth 2.0 consumer secret

   .
2. Uses the information friom the previous step to log in to the single login server and, if the login succeeds:
3. Sends an HTTP GET request to the server URL: `https://`***`instance`***`.salesforce.com/services/data/v23.0/sobjects/`. This is equivalent to a calling `describeGlobal()` to retrieve a list of all available objects for the organization's data.
4. Sends an HTTP GET request to the server URL: `https://`***`instance`***`.salesforce.com/services/data/v23.0/sobjects/Merchandise__c/describe/`. This is equivalent to a calling `describeSObject()` to retrieve metadata (field list and object properties) for the specified object.
5. Sends an HTTP POST request to the server URL: `https://`***`instance`***`.salesforce.com/services/data/v23.0/sobjects/Merchandise__c/` passing a JSON object in the request body. This is equivalent to a calling `create()` to a record corresponding to the JSON object.
6. Sends an HTTP GET request to the server URL: `https://`***`instance`***`.salesforce.com/services/data/v23.0/query/?q=SELECT+Name+FROM+Merchandise__c`. This is equivalent to a calling `query()`, passing a simple query string ("`SELECT Name FROM Merchandise__c`"), and iterating through the returned `QueryResult`.

Java Sample Code

```
package com.example.sample.rest;

import java.awt.Desktop;
```

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URLEncoder;

import org.apache.http.Header;
import org.apache.http.HttpResponse;
import org.apache.http.StatusLine;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.entity.StringEntity;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicHeader;
import org.apache.http.params.BasicHttpParams;
import org.apache.http.params.HttpParams;

import com.google.gson.Gson;
import com.google.gson.JsonElement;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;

public class RestClient extends Object {
  private static BufferedReader reader =
      new BufferedReader(new InputStreamReader(System.in));
  private static String OAUTH_ENDPOINT = "/services/oauth2/token";
  private static String REST_ENDPOINT = "/services/data";
  UserCredentials userCredentials;
  String restUri;
  Header oauthHeader;
  Header prettyPrintHeader = new BasicHeader("X-PrettyPrint", "1");
  Gson gson;
  OAuth2Response oauth2Response;

  public static void main(String[] args) {
    RestClient client = new RestClient();
    client.oauth2Login( client.getUserCredentials() );
    client.testRestData();
  }

  public RestClient() {
    gson = new Gson();
  }

  public HttpResponse oauth2Login(UserCredentials userCredentials) {
    HttpResponse response = null;
    this.userCredentials = userCredentials;
    String loginHostUri = "https://" +
        userCredentials.loginInstanceDomain + OAUTH_ENDPOINT;
    try {
      HttpClient httpClient = new DefaultHttpClient();
      HttpPost httpPost = new HttpPost(loginHostUri);
      StringBuffer requestBodyText =
          new StringBuffer("grant_type=password");
      requestBodyText.append("&username=");
      requestBodyText.append(userCredentials.userName);
      requestBodyText.append("&password=");
      requestBodyText.append(userCredentials.password);
      requestBodyText.append("&client_id=");
      requestBodyText.append(userCredentials.consumerKey);
      requestBodyText.append("&client_secret=");
      requestBodyText.append(userCredentials.consumerSecret);
```

```
    StringEntity requestBody =
        new StringEntity(requestBodyText.toString());
    requestBody.setContentType("application/x-www-form-urlencoded");
    httpPost.setEntity(requestBody);
    httpPost.addHeader(prettyPrintHeader);
    response = httpClient.execute(httpPost);
    if (  response.getStatusLine().getStatusCode() == 200 ) {
      InputStreamReader inputStream = new InputStreamReader(
          response.getEntity().getContent()
      );
      oauth2Response = gson.fromJson( inputStream,
          OAuth2Response.class );
      restUri = oauth2Response.instance_url + REST_ENDPOINT +
          "/v" + this.userCredentials.apiVersion +".0";
      System.out.println("\nSuccessfully logged in to instance: " +
          restUri);
      oauthHeader = new BasicHeader("Authorization", "OAuth " +
          oauth2Response.access_token);
    } else {
      System.out.println("An error has occured.");
      System.exit(-1);
    }
  } catch (UnsupportedEncodingException uee) {
    uee.printStackTrace();
  } catch (IOException ioe) {
    ioe.printStackTrace();
  } catch (NullPointerException npe) {
    npe.printStackTrace();
  }
  return response;
}

public void testRestData() {
  String responseBody = restGet(restUri);
  responseBody = restGet(restUri + "/sobjects/");
  responseBody = restGet(restUri +
      "/sobjects/Merchandise__c/describe/");
  responseBody = restPost(restUri +
      "/sobjects/Merchandise__c/", "{ \"Name\" : \"Wee Jet\" }\n\n");
  System.out.println(responseBody);
  JsonParser jsonParser = new JsonParser();
  JsonElement jsonElement = jsonParser.parse(responseBody);
  String id = jsonElement.getAsJsonObject().get("id").getAsString();
  responseBody = restGet(restUri +
      "/sobjects/Merchandise__c/" + id);
  System.out.println(responseBody);
  responseBody = restPost(restUri +
      "/sobjects/Merchandise__c/", "{ \"Name\" : \"Zeppelin GmbH\" }\n\n");
  System.out.println(responseBody);
  responseBody = restGet(restUri +
      "/query/?q=SELECT+Name+FROM+Merchandise__c");
  System.out.println(responseBody);
  responseBody = restPatch(restUri +
      "/sobjects/Merchandise__c/" + id, "{ \"Name\" : \"Dry Twig.\" }\n\n");
  System.out.println(responseBody);
  responseBody = restGet(restUri +
      "/sobjects/Merchandise__c/" + id);
  System.out.println(responseBody);
}

public String restGet(String uri) {
  String result = "";
  printBanner("GET", uri);
  try {
    HttpClient httpClient = new DefaultHttpClient();
    HttpGet httpGet = new HttpGet(uri);
    httpGet.addHeader(oauthHeader);
```

```
      httpGet.addHeader(prettyPrintHeader);
      HttpResponse response = httpClient.execute(httpGet);
      result = getBody( response.getEntity().getContent() );
    } catch (IOException ioe) {
      ioe.printStackTrace();
    } catch (NullPointerException npe) {
      npe.printStackTrace();
    }
    return result;
 }

 public String restPatch(String uri, String requestBody) {
    String result = "";
    printBanner("PATCH", uri);
    try {
      HttpClient httpClient = new DefaultHttpClient();
      HttpPatch httpPatch = new HttpPatch(uri);
      httpPatch.addHeader(oauthHeader);
      httpPatch.addHeader(prettyPrintHeader);
      StringEntity body = new StringEntity(requestBody);
      body.setContentType("application/json");
      httpPatch.setEntity(body);
      HttpResponse response = httpClient.execute(httpPatch);
      result = response.getEntity() != null ?
          getBody( response.getEntity().getContent() ) : "";
    } catch (IOException ioe) {
      ioe.printStackTrace();
    } catch (NullPointerException npe) {
      npe.printStackTrace();
    }
    return result;
}

 public String restPatchXml(String uri, String requestBody) {
    String result = "";
    printBanner("PATCH", uri);
    try {
      HttpClient httpClient = new DefaultHttpClient();
      HttpPatch httpPatch = new HttpPatch(uri);
      httpPatch.addHeader(oauthHeader);
      httpPatch.addHeader(prettyPrintHeader);
      httpPatch.addHeader( new BasicHeader("Accept", "application/xml") );
      StringEntity body = new StringEntity(requestBody);
      body.setContentType("application/xml");
      httpPatch.setEntity(body);
      HttpResponse response = httpClient.execute(httpPatch);
      result = getBody( response.getEntity().getContent() );
    } catch (IOException ioe) {
      ioe.printStackTrace();
    } catch (NullPointerException npe) {
      npe.printStackTrace();
    }
    return result;
}

 public String restPost(String uri, String requestBody) {
    String result = null;
    printBanner("POST", uri);
    try {
      HttpClient httpClient = new DefaultHttpClient();
      HttpPost httpPost = new HttpPost(uri);
      httpPost.addHeader(oauthHeader);
      httpPost.addHeader(prettyPrintHeader);
      StringEntity body = new StringEntity(requestBody);
      body.setContentType("application/json");
      httpPost.setEntity(body);
      HttpResponse response = httpClient.execute(httpPost);
```

```
      result = getBody( response.getEntity().getContent() );
    } catch (IOException ioe) {
      ioe.printStackTrace();
    } catch (NullPointerException npe) {
      npe.printStackTrace();
    }
    return result;
 }

  /**
   * Extend the Apache HttpPost method to implement an HttpPost
   * method.
   */
  public static class HttpPatch extends HttpPost {
    public HttpPatch(String uri) {
      super(uri);
    }

    public String getMethod() {
      return "PATCH";
    }
  }

  static class OAuth2Response {
    public OAuth2Response() {
    }
    String id;
    String issued_at;
    String instance_url;
    String signature;
    String access_token;
  }

  class UserCredentials {
    String grantType;
    String userName;
    String password;
    String consumerKey;
    String consumerSecret;
    String loginInstanceDomain;
    String apiVersion;
  }

  // private methods

  private String getUserInput(String prompt) {
    String result = "";
    try {
      System.out.print(prompt);
      result = reader.readLine();
    } catch (IOException ioe) {
      ioe.printStackTrace();
    }
    return result;
  }

  private void printBanner(String method, String uri) {

System.out.println("\n--------------------------------------------------------------------------------\n");

    System.out.println("HTTP Method: " + method);
    System.out.println("REST URI: " + uri);

System.out.println("\n--------------------------------------------------------------------------------\n");

  }
```

```
  private String getBody(InputStream inputStream) {
    String result = "";
    try {
        BufferedReader in = new BufferedReader(
            new InputStreamReader(inputStream)
        );
        String inputLine;
        while ( (inputLine = in.readLine() ) != null ) {
          result += inputLine;
          result += "\n";
        }
        in.close();
    } catch (IOException ioe) {
      ioe.printStackTrace();
    }
    return result;
  }

  private UserCredentials getUserCredentials() {
    UserCredentials userCredentials = new UserCredentials();
    userCredentials.loginInstanceDomain =
        getUserInput("Login Instance Domain: ");
    userCredentials.apiVersion = getUserInput("API Version: ");
    userCredentials.userName = getUserInput("UserName: ");
    userCredentials.password = getUserInput("Password: ");
    userCredentials.consumerKey = getUserInput("Consumer Key: ");
    userCredentials.consumerSecret = g
        etUserInput("Consumer Secret: ");
    userCredentials.grantType = "password";
    return userCredentials;
  }
}
```

See Also:

Step 4: Walk Through the Sample Code

USING REST RESOURCES

Chapter 6

Working with Objects and Records

In this chapter ...

- List Available REST API Versions
- List Available REST Resources
- Get a List of Objects
- Retrieve Metadata for an Object
- Get Field and Other Metadata for an Object
- Get Field Values from Records
- Execute a SOQL Query
- Search for a String
- Create a Record
- Update a Record
- Insert or Update (Upsert) a Record
- Delete a Record

The examples in this section explain how to use REST Resources to perform tasks such as creating, updating, or deleting a record.

For complete reference, see Reference on page 28.

List Available REST API Versions

Lists summary information about each REST API version currently available, including the version, label, and a link to each version's root. You do not need authentication to retrieve the list of versions.

Example usage

```
curl http://na1.salesforce.com/services/data/
```

Example request body

none required

Example response body

```
[
    {
        "label":"Winter '10"
        "version":"20.0",
        "url":"/services/data/v20.0",
    }
]
```

List Available REST Resources

List the resources available for the specified API version. It provides the name and URI of each resource.

Example

```
curl https://na1.salesforce.com/services/data/v20.0/ -H "Authorization: OAuth token"
 -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
{
    "sobjects" : "/services/data/v20.0/sobjects",
    "search" : "/services/data/v20.0/search",
    "query" : "/services/data/v20.0/query",
    "recent" : "/services/data/v20.0/recent"
}
```

Get a List of Objects

List the objects that are available in your organization and available to the logged-in user. This request also returns the organization encoding as well as maximum batch size permitted in queries.

Example usage

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/ -H "Authorization:
OAuth token" -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
Date: Thu, 21 Oct 2010 22:48:18 GMT
Transfer-Encoding: chunked
Content-Type: application/json;
charset=UTF-8 Server:
{
 "encoding" : "UTF-8",
 "maxBatchSize" : 200,
 "sobjects" : [ {
    "name" : "Widget__c",
    "updateable" : true,
    "label" : "Widget",
    "keyPrefix" : "a01",
    "custom" : true,
    "searchable" : true,
    "labelPlural" : "Widgets",
    "layoutable" : true,
    "activateable" : false,
    "urls" : { "sobject" : "/services/data/v20.0/sobjects/Widget__c",
    "describe" : "/services/data/v20.0/sobjects/Widget__c/describe",
    "rowTemplate" : "/services/data/v20.0/sobjects/Widget__c/{ID}" },
    "createable" : true,
    "customSetting" : false,
    "deletable" : true,
    "deprecatedAndHidden" : false,
    "feedEnabled" : false,
    "mergeable" : false,
    "queryable" : true,
    "replicateable" : true,
    "retrieveable" : true,
    "undeletable" : true,
    "triggerable" : true },
   },
...
```

Retrieve Metadata for an Object

Retrieve metadata for an object using the HTTP GET method.

Example for retrieving Widget custom object metadata

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/ -H
"Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body for retrieving Widget custom object metadata

none required

Example response body for retrieving Widget custom object metadata

```
{
    "objectDescribe" :
    {
        "name" : "Widget__c",
        "updateable" : true,
        "label" : "Widget",
        "keyPrefix" : "a01",
        "custom" : true,

        ...

        "replicateable" : true,
        "retrieveable" : true,
        "undeletable" : true,
        "triggerable" : true
    },
    "recentItems" : [ ]
}
```

To get a complete description of an object, including field names and their metadata, see Get a List of Objects on page 18.

Get Field and Other Metadata for an Object

Retrieve all the metadata for an object, including information about each field, URLs, and child relationships.

Example

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/describe/ -H
 "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
{
    "name" : "Widget__c",
    "fields" :
    [
        {
            "length" : 18,
            "name" : "Id",
```

```
            "type" : "id",
            "defaultValue" : {    "value" : null  },
            "updateable" : false,
            "label" : "Record ID",
            ...
        },

        ...

    ],


    "updateable" : true,
    "label" : "Widget",
    "keyPrefix" : "a01",
    "custom" : true,

    ...

    "urls" :
    {
        "uiEditTemplate" : "https://na1.salesforce.com/{ID}/e",
        "sobject" : "/services/data/v20.0/sobjects/Widget__c",
        "uiDetailTemplate" : "https://na1.salesforce.com/{ID}",
        ...
    },

    "childRelationships" :
    [
        {
            "field" : "FirstPublishLocationId",
            "deprecatedAndHidden" : false,
            ...
        },

        ....

    ],

    "createable" : true,
    "customSetting" : false,
    ...
}
```

Get Field Values from Records

Retrieve the value from fields on a record using the HTTP GET.

Example for retrieving values from fields on a Widget custom object

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/a01D000000INjVe
?fields=Name,Widget_Cost__c -H "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body

None required

Example response body

```
{
    "attributes" :
    {
        "type" : "Widget__c",
        "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000INjVe"
    }
    "Name" : "Test",
    "Widget_Cost__c" : 5.75,
    "Id" : "a01D000000INjVe"
}
```

Execute a SOQL Query

You can execute a SOQL query that returns all the results in a single response, or if needed, returns part of the results and an identifier used to retrieve the remaining results.

Example usage for executing a query

The following query requests the value from `name` fields from all Widget records.

```
curl https://na1.salesforce.com/services/data/v20.0/query/?q=SELECT+name+from+Widget__c
 -H "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body for executing a query

none required

Example response body for executing a query

```
{
    "done" : true,
    "totalSize" : 14,
    "records" :
    [
        {
            "attributes" :
            {
                "type" : "Widget__c",
                "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IRFmaIAH"

            },
            "Name" : "Test 1"
        },
        {
            "attributes" :
            {
                "type" : "Widget__c",
                "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IomazIAB"

            },
            "Name" : "Test 2"
        },

        ...
```

```
    ]
}
```

Retrieving the Remaining SOQL Query Results

If the initial query returns only part of the results, the end of the response will contain a field called `nextRecordsUrl`. For example, you might find this attribute at the end of your query:

```
"nextRecordsUrl" : "/services/data/v20.0/query/01gD0000002HU6KIAW-2000"
```

In such cases, request the next batch of records and repeat until all records have been retrieved. These requests use `nextRecordsUrl`, and do not include any parameters.

Example usage for retrieving the remaining query results

```
curl https://na1.salesforce.com/services/data/v20.0/query/01gD0000002HU6KIAW-2000 -H
 "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body for retrieving the remaining query results

none required

Example response body for retrieving the remaining query results

```
{
    "done" : true,
    "totalSize" : 3214,
    "records" : [...]
}
```

Search for a String

Execute the specified SOSL search. The search string, in this example `{test}`, must be URL-encoded.

Example usage

```
curl https://na1.salesforce.com/services/data/v20.0/search/?q=FIND+%7Btest%7D -H
"Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
[
    {
        "attributes" :
         {
```

```
            "type" : "Widget__c",
            "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IqhSLIAZ"
        },
        "Id" : "a01D000000IqhSLIAZ"
    },
    {
        "attributes" :
        {
            "type" : "Widget__c",
            "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IomazIAB"
        },
        "Id" : "a01D000000IomazIAB"
    }
]
```

Create a Record

Creating a Widget Record

Use the HTTP POST method to create a new Widget record.

Example cURL command

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/ -H
"Content-Type: application/json" -d @newwidget.json -H "Authorization: OAuth token"
-H "X-PrettyPrint:1"
```

Example request body for posting a new Widget (contents of `newwidget.json`)

```
{
    "Name" : "test"
}
```

Example response body for posting a new Widget

```
{
    "id" : "a01D000000IqhSLIAZ",
    "errors" : [ ],
    "success" : true
}
```

Error responses

See Error Response on page 43.

Creating Records with PATCH

The following example shows how to use PATCH with Apache's HttpClient, even though there is not yet a PatchMethod class in HttpClient.

```
public static void patch(String url, String sid) throws IOException {
 PostMethod m = new PostMethod(url) {
```

```
   @Override public String getName() { return "PATCH"; }
  };

  m.setRequestHeader("Authorization", "OAuth " + sid);

  Map<String, Object> accUpdate = new HashMap<String, Object>();
  accUpdate.put("Name", "Patch test");
  accUpdate.put("AnnualRevenue", 10);
  ObjectMapper mapper = new ObjectMapper();
  m.setRequestEntity(new StringRequestEntity(mapper.writeValueAsString(accUpdate),
"application/json", "UTF-8"));

  HttpClient c = new HttpClient();
  int sc = c.executeMethod(m);
  System.out.println("PATCH call returned a status code of " + sc);
  if (sc > 299) {
   // deserialize the returned error message
   List<ApiError> errors = mapper.readValue(m.getResponseBodyAsStream(), new
TypeReference<List<ApiError>>() {} );
   for (ApiError e : errors)
    System.out.println(e.errorCode + " " + e.message);
  }
 }

 private static class ApiError {
  public String errorCode;
  public String message;
  public String [] fields;
 }
```

This example uses the Jackson JSON library, especially useful for Java clients.

If you use an HTTP library that doesn't allow overriding or setting an arbitrary HTTP method name, you can send a POST request and provide an override to the HTTP method via the query string parameter `_HttpMethod`. In the PATCH example, you can replace the PostMethod line with one that doesn't use override:

```
PostMethod m = new PostMethod(url + "?_HttpMethod=PATCH");
```

Update a Record

Update a record using HTTP PATCH. First, create a file that contains the changes, then specify it in the cURL command.Records in a single file must be of the same object type.

Example for updating two fields in a Widget custom object

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/a01D000000INjVe
  -H "Authorization: OAuth token" -H "X-PrettyPrint:1" -H "Content-Type:
application/json" -d @patchwidget.json -X PATCH
```

Example request body for updating custom field in a Widget custom object

```
{
    "Widget_Cost__c" : 5.75
}
```

Example response body for updating fields in a Widget custom object

none returned

Error response

See Error Response on page 43.

Insert or Update (Upsert) a Record

Create new records or update existing records (upsert) based on the value of a specified external ID field.

- If the specified value doesn't exist, a new record is created.
- If a record does exist with that value, the field values specified in the request body are updated.
- If the value is not unique, the REST API returns a 300 response with the list of matching records.

The following sections show how to work with the extneral ID resource to retrieve records by external ID and upsert records, using either JSON or XML.

Upserting Existing Records with JSON

Use the PATCH method to update individual records, or create them if they don't already exist, based on the external ID field, for an object. In this example, the Widget Cost custom field on a Widget record is updated.

Example: upserting a record that already exists

```
curl
https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/External_Name__c/ExistingName
 -v –H "Authorization:
OAuth token" -H "Content-Type: application/json" -X PATCH -d @fileName
```

JSON Example request body for updating the `Widget Cost` field

Request body:

```
{

    Widget_Cost__c : 5.75

}
```

`Widget_Cost__c` is a custom field on the record, with the new value to be updated or inserted.

JSON Example Response

HTTP status code 204 is returned if an existing record is updated.

Error responses

If the external ID value isn't unique, an HTTP status code 300 is returned, plus a list of the records that matched the query. For more information about errors, see Error Response on page 43.

If the external ID field doesn't exist, an error message and code is returned:

```
{
  "message" : "The requested resource does not exist",
```

```
  "errorCode" : "NOT_FOUND"
}
```

Upserting Existing Records with XML

Example: upserting a record that already exists

```
curl
https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/External_Name__c/ExistingName
 -v –H "Authorization:
OAuth token" -H "Content-Type: application/xml" -X PATCH -d @widget.xml
```

Contents for file `widget.xml`, updating the `Name` and `Widget_Cost__c` fields

```
<root>
   <Name>Updated Name</Name>
   <Widget_Cost__c>4.45</Widget_Cost__c>
</root>
```

XML Example Response

HTTP status code 204 is returned if an existing record is updated.

XML Example Response

No response body with HTTP status code 204.

Error responses

Incorrect ID field:

```
<Errors>
    <Error>
        <errorCode>MALFORMED_ID</errorCode>
        <fields>Id</fields>
        <message>Record ID: id value of incorrect type: a00D0000008o5VtIAP</message>
    </Error>
</Errors>
```

For more error information, see Error Response on page 43.

Delete a Record

Delete a record using HTTP DELETE.

Use the DELETE method to delete an object.

Example for deleting a Widget record

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/a01D000000INjVe
  -H "Authorization: OAuth token" -H "X-PrettyPrint:1" -H "Authorization: OAuth token"
 -H "X-PrettyPrint:1" -X DELETE
```

Example request body

None needed

Example response body

None returned

Chapter 7

Reference

The table lists supported REST resources in the API in alphabetical order, and provides a brief description for each. In each case, the URI for the resource follows the base URI, which you retrieve from the authentication service: `http://domain/services/data`. ***domain*** might be the Salesforce instance you are using, or a custom domain.The domain is returned as part of the Authentication response, or via the Console at **Company Profile** > **My Domain**. For example, to retrieve basic information about a Widget custom object in version 20.0: `http://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/`.

Click a call name to see syntax, usage, and more information for that call.

Resource Name	URI	Description
Versions	`/`	Lists summary information about each Database.com version currently available, including the version, label, and a link to each version's root.
Resources by Version	`/vXX.X/`	Lists available resources for the specified API version, including resource name and URI.
Describe Global	`/vXX.X/sobjects/`	Lists the available objects and their metadata for your organization's data.
SObject Basic Information	`/vXX.X/sobjects/`***SObject***`/`	Describes the individual metadata for the specified object.
SObject Describe	`/vXX.X/sobjects/`***SObject***`/describe/`	Completely describes the individual metadata at all levels for the specified object.
SObject Rows	`/vXX.X/sobjects/`***SObject***`/`***id***`/`	Accesses records based on the specified object ID. Retrieves, creates, updates, or deletes records.
SObject Rows by External ID	`/vXX.X/sobjects/`***SObjectName***`/`***fieldName***`/`***fieldValue***	Creates new records or updates existing records (upserts records) based on the value of a specified external ID field.
Query	`/vXX.X/query/?q=`***soql***	Executes the specified SOQL query.
Search	`/vXX.X/search/?s=`***sosl***	Executes the specified SOSL search. The search string must be URL-encoded.

Versions

Lists summary information about each Database.com version currently available, including the version, label, and a link to each version's root.

URI

/

Formats

JSON, XML

HTTP Method

GET

Authentication

none

Parameters

none

Example usage

```
curl https://na1.salesforce.com/services/data/
```

Example request body

none required

Example response body

```
[
    {
        "label":"Winter '10"
        "version":"20.0",
        "url":"/services/data/v20.0",
    }
]
```

Resources by Version

Lists available resources for the specified API version, including resource name and URI.

URI

/vXX.X/

Formats

JSON, XML

HTTP Method

GET

Authentication

```
Authorization: OAuth token
```

Parameters

none

Example

```
curl https://na1.salesforce.com/services/data/v20.0/ -H "Authorization: OAuth token"
 -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
{
    "sobjects" : "/services/data/v20.0/sobjects",
    "search" : "/services/data/v20.0/search",
    "query" : "/services/data/v20.0/query",
    "recent" : "/services/data/v20.0/recent"
}
```

Describe Global

Lists the available objects and their metadata for your organization's data. In addition, it provides the organization encoding, as well as maximum batch size permitted in queries. For more information, see Internationalization and Character Sets.

URI

```
/vXX.X/sobjects/
```

Formats

JSON, XML

HTTP Method

GET

Authentication

```
Authorization: OAuth token
```

Parameters

none required

Example usage

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/ -H "Authorization:
OAuth token" -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
{
    "encoding" : "UTF-8",
    "maxBatchSize" : 200,
    "sobjects" : [ {
       "name" : "Widget__c",
       "label" : "Widget",
       "custom" : true,
       "keyPrefix" : "a01",
       "updateable" : true,
       "searchable" : true,
       "labelPlural" : "Widgets",
       "layoutable" : true,
       "activateable" : false,
       "urls" : {
         "sobject" : "/services/data/v20.0/sobjects/Widget__c",
         "describe" : "/services/data/v20.0/sobjects/Widget__c/describe",
         "rowTemplate" : "/services/data/v20.0/sobjects/Widget__c/{ID}" },
       },
       "createable" : true,
       "customSetting" : false,
       "deletable" : true,
       "deprecatedAndHidden" : false,
       "feedEnabled" : false,
       "mergeable" : false,
       "queryable" : true,
       "replicateable" : true,
       "retrieveable" : true,
       "undeletable" : true,
       "triggerable" : true
    } ]
}
```

Error responses

See Error Response on page 43.

SObject Basic Information

Describes the individual metadata for the specified object. For example, this can be used to retrieve the metadata for a custom object or post a new custom object.

URI

`/vXX.X/sobjects/SObjectName/`

Formats

JSON, XML

HTTP Method

GET, POST

Authentication

```
Authorization: OAuth token
```

Parameters

none required

Retrieving Custom Object Metadata

Use the GET method to retrieve the metadata for an object. In this example, the Widget custom object's metadata are retrieved.

Example usage for retrieving Widget metadata

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/ -H
"Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body for retrieving Widget metadata

none required

Example response body for retrieving Widget metadata

```
{
    "objectDescribe" :
    {
        "name" : "Widget__c",
        "updateable" : true,
        "label" : "Widget",
        "keyPrefix" : "a01",

        ...

        "replicateable" : true,
        "retrieveable" : true,
        "undeletable" : true,
        "triggerable" : true
    },
    "recentItems" : [ ]
}
```

Error responses

See Error Response on page 43.

Creating a New Custom Object Record

Use the POST method to create a new object. In this example, a new Widget custom object record is created.

Example usage for posting a new Widget custom object

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/ -H
"Content-Type: application/json" -d @newwidget.json -H "Authorization: OAuth token"
-H "X-PrettyPrint:1"
```

Example request body for posting a new Widget

```
{
    "Name" : "test"
}
```

Example response body for posting a new Widget

```
{
    "id" : "a01D000000IqhSLIAZ",
    "errors" : [ ],
    "success" : true
}
```

Example error response

```
{
  "message" : "Cannot deserialize instance of currency from VALUE_STRING value test_string
 at [Source: HTTP Body; line: 1, column: 69]",
  "errorCode" : "JSON_PARSER_ERROR"
}
```

For more information about error codes, see Error Response on page 43.

SObject Describe

Completely describes the individual metadata at all levels for the specified object. For example, this can be used to retrieve the fields, URLs, and child relationships for a custom object.

URI

`/vXX.X/sobjects/SObjectName/describe/`

Formats

JSON, XML

HTTP Method

GET

Authentication

`Authorization: OAuth token`

Parameters

none required

Example usage

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/describe/ -H
 "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
{
    "name" : "Widget__c",
    "fields" :
    [
        {
            "length" : 18,
            "name" : "Id",
            "type" : "id",
            "defaultValue" : null,
            "updateable" : false,
            "label" : "Record ID",
            ...
        },

        ...

    ],


    "updateable" : true,
    "label" : "Widget",
    "keyPrefix" : "a01",
    "custom" : true,

    ...

    "urls" :
    {
        "uiEditTemplate" : "https://na1.salesforce.com/{ID}/e",
        "sobject" : "/services/data/v20.0/sobjects/Widget__c",
        "uiDetailTemplate" : "https://na1.salesforce.com/{ID}",
        ...
    },

    "childRelationships" :
    [
        {
            "field" : "ParentId",
            "deprecatedAndHidden" : false,
            ...
        },

        ....

    ],

    "createable" : true,
    "customSetting" : false,
```

```
    ...
}
```

SObject Rows

Accesses records based on the specified object ID. Retrieves, creates, updates, or deletes records.

URI

`/vXX.X/sobjects/`***`SObjectName`***`/`***`id`***`/`

Formats

JSON, XML

HTTP Method

GET, PATCH, DELETE

Authentication

`Authorization: OAuth` ***`token`***

Parameters

Parameter	Description
`fields`	Optional list of fields used to return values for.

Retrieving Fields from a Widget Custom Object

Use the GET method to retrieve individual records for an object. In this example, the name and cost are retrieved from a Widget custom object.

Example usage for retrieving fields from a Widget custom object

```
curl
https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/a01D000000INjVe?fields=Name,Widget_Cost__c
 -H "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body for retrieving fields from a Widget custom object

none required

Example response body for retrieving fields from a Widget custom object

```
{
    "Name" : "Test",
    "Widget_Cost__c" : 5.75,
    "Id" : "a01D000000INjVe"
}
```

Error responses

See Error Response on page 43.

Updating Fields on a Widget Custom Object

Use the PATCH method to update individual records on an object. In this example, the `Widget Cost` within a Widget is updated.

Example usage for updating fields in a Widget custom object

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/a01D000000INjVe
  -H "Authorization: OAuth token" -H "X-PrettyPrint:1" -H "Content-Type:
application/json" -d @patchwidget.json -X PATCH
```

Example request body for updating fields in a Widget custom object

```
{
    "Widget_Cost__c" : 5.75
}
```

Example response body for updating fields in a Widget custom object

none returned

Error responses

See Error Response on page 43.

Creating a Widget Record

Use the HTTP POST method to create a new Widget record.

Example cURL command

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/ -H
"Content-Type: application/json" -d @newwidget.json -H "Authorization: OAuth token"
-H "X-PrettyPrint:1"
```

Example request body for posting a new Widget (contents of `newwidget.json`)

```
{
    "Name" : "test"
}
```

Example response body for posting a new Widget

```
{
    "id" : "a01D000000IqhSLIAZ",
    "errors" : [ ],
    "success" : true
}
```

Error responses

See Error Response on page 43.

Deleting a Widget Record

Use the DELETE method to delete a record. In this example, a Widget record is deleted.

Example usage for deleting fields in a Widget object

```
curl https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/a01D000000INjVe
  -H "Authorization: OAuth token" -H "X-PrettyPrint:1" -X DELETE
```

Example request body for deleting a Widget record

none required

Example response body for deleting a Widget record

none returned

SObject Rows by External ID

Creates new records or updates existing records (upserts records) based on the value of a specified external ID field.

- If the specified value doesn't exist, a new record is created.
- If a record does exist with that value, the field values specified in the request body are updated.
- If the value is not unique, the REST API returns a 300 response with the list of matching records.

Note: Do not specify `Id` or an external ID field in the request body or an error is generated.

URI

`/vXX.X/sobjects/`***`SObjectName`***`/`***`fieldName`***`/`***`fieldValue`***

Formats

JSON, XML

HTTP Method

HEAD, GET, PATCH, DELETE

Authentication

`Authorization: OAuth` ***`token`***

Parameters

None

Upserting Existing Records with JSON

Use the PATCH method to update individual records, or create them if they don't already exist, based on the external ID field, for an object. In this example, the Widget Cost custom field on a Widget record is updated.

Example: upserting a record that already exists

```
curl
https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/External_Name__c/ExistingName
 -v –H "Authorization:
OAuth token" -H "Content-Type: application/json" -X PATCH -d @fileName
```

JSON Example request body for updating the `Widget Cost` field

Request body:

```
{

    Widget_Cost__c : 5.75

}
```

`Widget_Cost__c` is a custom field on the record, with the new value to be updated or inserted.

JSON Example Response

HTTP status code 204 is returned if an existing record is updated.

Error responses

If the external ID value isn't unique, an HTTP status code 300 is returned, plus a list of the records that matched the query. For more information about errors, see Error Response on page 43.

If the external ID field doesn't exist, an error message and code is returned:

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

Upserting New Records with JSON

Example: upserting a record that does not exist yet

```
curl
https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/External_Name__c/NewName
 -v –H "Authorization:
OAuth token" -H "Content-Type: application/json" -X "PATCH" -d @newwidget.json
```

Contents for file `newwidget.json`, updating the `Name` and `Widget_Cost__c` fields

```
{

 "Name" : "A New Widget",
 "Widget_Cost__c" : 6.75

}
```

Response

Successful response:

```
{
  "id" : "a01D0000001pPvHAAU",
  "errors" : [ ],
  "success" : true
}
```

The response body is empty. HTTP status code 201 is returned if a new record is created.

Error responses

Incorrect external ID field:

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

For more information, see Error Response on page 43.

Upserting Existing Records with XML

Example: upserting a record that already exists

```
curl
https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/External_Name__c/ExistingName
 -v –H "Authorization:
OAuth token" -H "Content-Type: application/xml" -X PATCH -d @widget.xml
```

Contents for file `widget.xml`, updating the `Name` and `Widget_Cost__c` fields

```
<root>
   <Name>Updated Name</Name>
   <Widget_Cost__c>4.45</Widget_Cost__c>
</root>
```

XML Example Response

HTTP status code 204 is returned if an existing record is updated.

XML Example Response

No response body with HTTP status code 204.

Error responses

Incorrect ID field:

```
<Errors>
    <Error>
        <errorCode>MALFORMED_ID</errorCode>
        <fields>Id</fields>
        <message>Record ID: id value of incorrect type: a00D0000008o5VtIAP</message>
    </Error>
</Errors>
```

For more error information, see Error Response on page 43.

Upserting New Records with XML

XML Example: upserting a new record

```
curl
https://na1.salesforce.com/services/data/v20.0/sobjects/Widget__c/External_Name__c/NewName
 -v –H "Authorization:
OAuth token" -H "Content-Type: application/xml" -X PATCH -d @widget.xml
```

Contents for file `widget.xml`, updating the `Name` and `Widget_Cost__c` fields

```
<root>
   <Name>A New Widget</Name>
   <Widget_Cost__c>8.23</Widget_Cost__c>
</root>
```

XML Example Response

No response body with HTTP status code 204.

Error response

Incorrect ID field:

```
<Errors>
    <Error>
        <errorCode>MALFORMED_ID</errorCode>
        <fields>Id</fields>
        <message>Record ID: id value of incorrect type: a00D0000008o5VtIAP</message>
    </Error>
</Errors>
```

For more information, see Error Response on page 43.

Query

Executes the specified SOQL query.

URI

`/vXX.X/query/?q=soql`

Formats

JSON, XML

HTTP Method

GET

Authentication

`Authorization: OAuth token`

Parameters

Parameter	Description
q	A SOQL query.

Executing the SOQL Query

Execute a SOQL query that returns all the results in a single response, or if there are too many results, returns the first part of the results and an identifier used to retrieve the remaining results.

Example query

```
curl https://na1.salesforce.com/services/data/v20.0/query/?q=SELECT+name+from+Widget__c
 -H "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body for executing a query

none required

Example response body for executing a query

```
{
    "done" : true,
    "totalSize" : 14,
    "records" :
    [
        {
            "attributes" :
            {
                "type" : "Widget__c",
                "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IRFmaIAH"

            },
            "Name" : "Test 1"
        },
        {
            "attributes" :
            {
                "type" : "Widget__c",
                "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IomazIAB"

            },
            "Name" : "Test 2"
        },

        ...

    ]
}
```

Error responses

See Error Response on page 43.

Retrieving the Remaining SOQL Query Results

If the initial query returns only part of the results, the end of the response will contain a field called `nextRecordsUrl`. For example, you might find this attribute at the end of your query:

```
"nextRecordsUrl" : "/services/data/v20.0/query/01gD0000002HU6KIAW-2000"
```

In such cases, request the next batch of records and repeat until all records have been retrieved. These requests use `nextRecordsUrl`, and do not include any parameters.

Example usage for retrieving the remaining query results

```
curl https://na1.salesforce.com/services/data/v20.0/query/01gD0000002HU6KIAW-2000 -H
 "Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body for retrieving the remaining query results

none required

Example response body for retrieving the remaining query results

```
{
    "done" : true,
    "totalSize" : 3214,
    "records" : [...]
}
```

Error responses

See Error Response on page 43.

Search

Executes the specified SOSL search. The search string must be URL-encoded.

URI

`/vXX.X/search/?q=sosl`

Formats

JSON, XML

HTTP Method

GET

Authentication

`Authorization: OAuth token`

Parameters

Parameter	Description
q	A SOSL statement.

Example usage

The search string, `{test}`, has been URL-encoded.

```
curl https://na1.salesforce.com/services/data/v20.0/search/?q=FIND+%7Btest%7D -H
"Authorization: OAuth token" -H "X-PrettyPrint:1"
```

Example request body

none required

Example response body

```
[
    {
        "attributes" :
         {
            "type" : "Widget__c",
            "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IqhSLIAZ"
        },
        "Id" : "a01D000000IqhSLIAZ"
    },
    {
        "attributes" :
        {
            "type" : "Widget__c",
            "url" : "/services/data/v20.0/sobjects/Widget__c/a01D000000IomazIAB"
        },
        "Id" : "a01D000000IomazIAB"
    }
]
```

Error Response

When errors occur, the response header contains an HTTP code, and the response body usually contains the error code, the message accompanying that error, and the field or object where the error occurred.

HTTP Response Code	Description
300	The value used for an external ID exists in more than one record. The response body contains the list of matching records.
400	The request could not be understood, usually because the JSON or XML body has an error

401	The session ID or OAuth token used has expired or is invalid. The response body contains the `message` and `errorCode`.
403	The request has been refused. Verify that the logged-in user has appropriate permissions.
404	The requested resource could not be found. Check the URI for errors, and verify that there are no sharing issues.
405	The method specified in the Request-Line is not allowed for the resource specified in the URI.
415	The entity specified in the request is in a format that is not supported by specified resource for the specified method.
500	An error has occurred within Force.com, so the request could not be completed. Contact salesforce.com Customer Support.

Incorrect ID example

Using a non-existent ID in a request using JSON or XML (*`request_body`*`.json` or *`request_body`*`.xml`)

```
{
  "fields" : [ ],
  "message" : "malformed id a01900K0001pPuOAAU",
  "errorCode" : "MALFORMED_ID"
}
```

Resource does not exist

Requesting a resource that does not exist, for example, you try to create a record using a misspelled object name

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

Index

B

base URI 4

C

Compression 5
    deflate 5
    gzip 5
Create 23

D

date-time 4
Delete record 26
Describe Global 18, 30

E

Error response 43

F

Field values 20
    retrieving values 20

J

JSON 4

L

List REST resources 17

O

OAuth 4
    OAuth 2.0 4
Object metadata retrieval 18

P

PATCH 23
    creating records with 23

Q

Query 21, 40

R

Resource list by version 29
Resources 25, 37
    SObject upsert 37
    upsert 25
REST 4
    architecture 4
    cache 4
    gateway 4
    proxy server 4
    resources 4
    stateless 4
REST API 3
REST resources 3–4, 28
    list of REST resources 28
REST resources list 17
Retrieve object metadata 18
Retrieving field values 20

S

Search 22, 42
SObject Basic Information 31
SObject Describe 19, 33
SObject Row 24, 35
SObject upsert 37

U

Upsert 25
Upsert, sObject 37

V

Versions 17, 29

X

XML 4