

1. [Lightning Platform API Basics](#)



2. [Use REST API](#)

Use REST API

Learning Objectives

After completing this unit, you'll be able to:

- Log in to Workbench and navigate to REST Explorer.
- Use the describe resource.
- Create an account using REST API.
- Execute a query using REST API.

REST Resources and Methods

Land ho! We've spotted the Isle of REST ahead of the bow, captain. Before we dock and start using the API, let's talk about REST resources and methods.

A REST resource is an abstraction of a piece of information or an action, such as a single data record, a collection of records, or a query. Each resource in REST API is identified by a named Uniform Resource Identifier (URI) and is accessed using standard HTTP methods (HEAD, GET, POST, PATCH, DELETE). REST API is based on the usage of resources, their URIs, and the links between them.

You use a resource to interact with your Salesforce org. For example, you can:

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

A REST request consists of four components: a resource URI, an HTTP method, request headers, and a request body. Request headers specify metadata for the request. The request body specifies data for the request, when necessary. If there's no data to specify, the body is omitted from the request.

Describe the Account Object

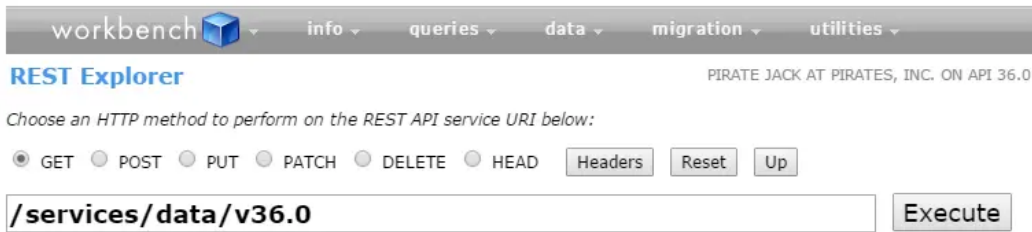
It's time to get our feet wet. We're going to use Workbench to make some API calls. Workbench is a suite of tools for interacting with your Salesforce org through the API. Because you can make REST requests from any HTTP sender, there are plenty of other tools available for you to use (for example, check out cURL or Postman). But because Workbench provides a friendly framework specifically for Salesforce APIs, it's the perfect way to dig in before you're ready to write a full-on integration.

The first step is to log in to Workbench.

1. Log in to your Trailhead Playground and navigate to [Workbench](#).
2. For Environment, select **Production**.
3. For API Version, select the highest available number.
4. Make sure that you select **I agree to the terms of service**.
5. Click **Login with Salesforce**.

You've arrived at the Workbench home page. For this module, we use only one of Workbench's many tools, the REST Explorer.

In the top menu, select **utilities** | **REST Explorer**.



workbench info queries data migration utilities

REST Explorer PIRATE JACK AT PIRATES, INC. ON API 36.0

Choose an HTTP method to perform on the REST API service URI below:

☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD Headers Reset Up

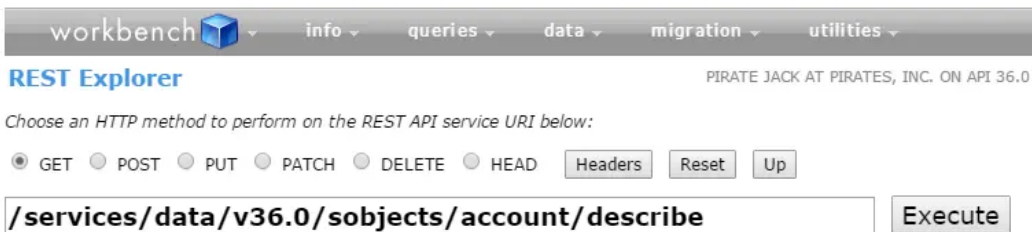
/services/data/v36.0 Execute

You can make REST API calls from the REST explorer just like you would from any other HTTP interface. The text in the text box represents a resource URI. For convenience, the top-level domain is omitted from the displayed URI. For example, the full URI of the resource that's prepopulated in the URI text box is <https://foo.my.salesforce.com/services/data/v36.0>.

The radio buttons above the URI represent the standard HTTP methods. To make an API call, enter the resource URI, select the appropriate method, add headers as needed, and click **Execute**.

Let's try out the SOBJect Describe resource. This resource, when combined with the GET method, returns metadata about an object and its fields.

We'll try describing the Account object. Replace the existing text in the URI text box with `/services/data/vXX.0/objects/account/describe`, where `XX` maps to the API version you're using.



workbench info queries data migration utilities

REST Explorer PIRATE JACK AT PIRATES, INC. ON API 36.0

Choose an HTTP method to perform on the REST API service URI below:


☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD Headers Reset Up

/services/data/v36.0/objects/account/describe Execute

Let's take a minute to break down this resource's URI.

- `/services/data` —Specifies that we're making a REST API request
- `/v36.0` —API version number
- `/objects` —Specifies that we're accessing a resource under the sObject grouping
- `/account` —sObject being actioned; in this case, account
- `/describe` —Action; in this case, a describe request

Now make sure that the GET method is selected, and click **Execute**.

workbench  info queries data migration utilities

REST Explorer

PIRATE JACK AT PIRATES, INC. ON API 36.0

Choose an HTTP method to perform on the REST API service URI below:


☒ GET
 ☐ POST
 ☐ PUT
 ☐ PATCH
 ☐ DELETE
 ☐ HEAD
 Headers Reset Up

/services/data/v36.0/subjects/account/describe Execute

Expand All | Collapse All | Show Raw Response

- actionOverrides
- activateable: **false**
- childRelationships
- compactLayoutable: **true**
- createable: **true**
- custom: **false**
- customSetting: **false**
- deletable: **true**
- deprecatedAndHidden: **false**
- feedEnabled: **true**
- fields
- keyPrefix: **001**
- label: **Account**
- labelPlural: **Accounts**
- layoutable: **true**
- listviewable: **null**
- lookupLayoutable: **null**
- mergeable: **true**
- name: **Account**
- namedLayoutInfos
- networkScopeFieldName: **null**
- queryable: **true**
- recordTypeInfo
- replicable: **true**
- retrieveable: **true**
- searchLayoutable: **true**
- searchable: **true**
- supportedScopes
- triggerable: **true**
- undeletable: **true**
- updateable: **true**
- urls

Good work, captain. The Account metadata appears on the screen. And Workbench has nicely formatted the response. To see the raw JSON response, click **Show Raw Response**.

workbench  info queries data migration utilities

REST Explorer

PIRATE JACK AT PIRATES, INC. ON API 36.0

Choose an HTTP method to perform on the REST API service URI below:

☒ GET
 ☐ POST
 ☐ PUT
 ☐ PATCH
 ☐ DELETE
 ☐ HEAD
 Headers Reset Up

/services/data/v36.0/subjects/account/describe Execute

Expand All | Collapse All | Hide Raw Response

- actionOverrides
- activateable: **false**
- childRelationships
- compactLayoutable: **true**
- createable: **true**
- custom: **false**
- customSetting: **false**
- deletable: **true**
- deprecatedAndHidden: **false**
- feedEnabled: **true**
- fields
- keyPrefix: **001**
- label: **Account**
- labelPlural: **Accounts**
- layoutable: **true**
- listviewable: **null**
- lookupLayoutable: **null**
- mergeable: **true**
- name: **Account**
- namedLayoutInfos
- networkScopeFieldName: **null**
- queryable: **true**
- recordTypeInfo
- replicable: **true**

Raw Response

```

HTTP/1.1 200 OK
Date: Mon, 16 May 2016 22:23:07 GMT
Set-Cookie:
BrowserId=SnOoG01nTqanUfUMSgJKg;Path=/;Domain=.salesforce.com;Expires=Fri, 15-Jul-2016 22:23:07 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Sforce-Limit-Info: api-usage=3/15000
org.eclipse.jetty.server.include.ETag: "120dfb8e"
Last-Modified: Mon, 16 May 2016 21:57:10 GMT
Content-Type: application/json;charset=UTF-8
Content-Encoding: gzip
ETag: "120dfb8e-gzip"
Transfer-Encoding: chunked

{
  "actionOverrides" : [ ],
  "activateable" : false,
  "childRelationships" : [ {
    "cascadeDelete" : false,
    "childSObject" : "Account",
    "deprecatedAndHidden" : false,
    "field" : "ParentId",
    "junctionIdListName" : null,
    "junctionReferenceTo" : [ ],
    "relationshipName" : "ChildAccounts",
    "restrictedDelete" : false
  } ], {

```

The Account metadata is displayed in JSON below some HTTP response headers. Because REST API supports both JSON and XML, let's change the request header to specify an XML response. Next to the HTTP methods, click **Headers**. For the **Accept** header value, replace `application/json` with `application/xml`. Your request headers look like this.

Request Headers

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

[Restore Default Headers](#)

Click **Execute**. The raw XML response is returned. Hurrah!

Create an Account

Now let's create an account using the SObject resource and the POST method. In the URI text box, replace the existing text with `/services/data/vXX.0/subjects/account`, where `XX` maps to the API version you're using. Select **POST**. Notice that a Request Body text area appears, which is where we specify the field values for our new account. First, though, let's change the **Accept** header back to JSON.

Click **Headers**. Change `Accept: application/xml` back to `Accept: application/json`. Your request looks like this.

workbench
info
queries
data
migration
utilities

REST Explorer

PIRATE JACK AT PIRATES, INC. ON API 36.0

Choose an HTTP method to perform on the REST API service URI below:

GET
POST
PUT
PATCH
DELETE
HEAD
Headers
Reset
Up

/services/data/v36.0/subjects/account

Execute

Request Headers

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

Restore Default Headers

Request Body

In the request body, enter the following text.

```
{
  "Name" : "NewAccount1",
  "ShippingCity" : "San Francisco"
}
```

Copy

Click **Execute**. You see a response such as the following.

workbench
info
queries
data
migration
utilities

REST Explorer

PIRATE JACK AT PIRATES, INC. ON API 36.0

Choose an HTTP method to perform on the REST API service URI below:

GET
POST
PUT
PATCH
DELETE
HEAD
Headers
Reset
Up

/services/data/v36.0/subjects/account

Execute

Request Body

```
{
  "Name" : "NewAccount1",
  "ShippingCity" : "San Francisco"
}
```

Expand All | Collapse All | Show Raw Response

id: 0013600000L01UiAAJ
success: true
errors

If `success: true`, the account was created with the returned ID. Expand the errors folder to check for errors. Just for kicks, let's create a second account without specifying an account name. Replace the text in the request body with the following text.

```
{
  "ShippingCity" : "San Francisco"
}
```

Copy

Click **Execute**.

Uh, oh. Did you get a `REQUIRED_FIELD_MISSING` response? Expand the **REQUIRED_FIELD_MISSING** folder, and then expand the **fields** folder. Your expanded response looks like this.

```
REQUIRED_FIELD_MISSING
  message: Required fields are missing: [Name]
  errorCode: REQUIRED_FIELD_MISSING
  fields
    0: Name
```

Because Name is a required field for creating an account, the server didn't process the request, and we received an error. Thankfully, all the information we need to fix the request is in the error response. Let's specify the name `NewAccount2`, and change the shipping city in the request body. Replace the text in the request body with the following text.

```
{
  "Name" : "NewAccount2",
  "ShippingCity" : "New York"
}
```

Copy

Click **Execute**. Success!

Execute a Query

Now let's imagine that you or another user has created hundreds of accounts. You want to find the names of all the accounts where the shipping city is San Francisco. You can use the Query resource to execute a SOQL query and zero in on the exact records you want, just like a customized treasure map.

Replace the text in the URI text box with the following text: `/services/data/vXX.0/query/?q=SELECT+Name+From+Account+WHERE+ShippingCity='San+Francisco'`, where `XX` maps to the API version you're using.

We replaced spaces with the `+` character in the query string to properly encode the URI. You can read about HTML URL encoding from the link in the Resources section. Make sure that the GET method is selected, and click **Execute**.

Expand the **records** folder. Do you see a folder with the name of the first account we created, `NewAccount1`? Great. Click it. Now click the **attributes** folder. Next to url is the resource URI of the account that was returned. Your response looks something like this.

```
totalSize: 1
done: true
records
  NewAccount1
    attributes
      type: Account
      url: /services/data/v36.0/subjects/Account/0013600000L01UiAAJ
      Name: NewAccount1
```

When you write an integration, you can grab this URI from the response to access more details about that account.

Node.js and Ruby Samples

Now you have a sweet taste of what's possible with REST API. Of course, when you move from Workbench to writing code, you'll be interacting with REST API using the programming language of your choice. Luckily, expert Salesforce developers have written wrappers for several languages that simplify the process of consuming REST API. Here are two sample queries written in Node.js and Ruby that use wrappers Nforce and Restforce, respectively.

Node.js Sample Using Nforce

```
var nforce = require('nforce');
// create the connection with the Salesforce connected app
var org = nforce.createConnection({
  clientId: process.env.CLIENT_ID,
  clientSecret: process.env.CLIENT_SECRET,
  redirectUri: process.env.CALLBACK_URL,
  mode: 'single'
});
// authenticate and return OAuth token
org.authenticate({
  username: process.env.USERNAME,
  password: process.env.PASSWORD+process.env.SECURITY_TOKEN
}, function(err, resp){
  if (!err) {
```

```
console.log('Successfully logged in! Cached Token: ' + org.oauth.access_token);
// execute the query
org.query({ query: 'select id, name from account limit 5' }, function(err, resp){
  if(!err && resp.records) {
    // output the account names
    for (i=0; i<resp.records.length;i++) {
      console.log(resp.records[i].get('name'));
    }
  }
});
if (err) console.log(err);
});
```

[Copy](#)

Ruby Sample Using Restforce

```
require 'restforce'
// create the connection with the Salesforce connected app
client = Restforce.new :username => ENV['USERNAME'],
  :password => ENV['PASSWORD'],
  :security_token => ENV['SECURITY_TOKEN'],
  :client_id => ENV['CLIENT_ID'],
  :client_secret => ENV['CLIENT_SECRET']
// execute the query
accounts = client.query("select id, name from account limit 5")
// output the account names
accounts.each do |account|
  p account.Name
end
```

[Copy](#)

Resources

[REST API Developer Guide](#)[Getting Started with Salesforce REST in Java](#)[HTML URL Encoding Reference](#)[Workbench](#)

Note

Remember, this module is meant for Salesforce Classic. When you launch your hands-on org, switch to Salesforce Classic to complete this challenge.

Assessment Complete!

+500 points



Lightning Platform API Basics

100%

Progress: 100%

Retake this Challenge

[View more modules](#)