

Lab Guide

ServiceNow custom REST APIs:
Build Custom Services the right
way with Scripted REST APIs

Jason McKee

Lab instance: <http://clabs.link/custom-api-pnq>

Default Login / Password:

admin / pnq-cc17

itil / pnq-cc17

employee / pnq-cc17

This
Page
Intentionally
Left
Blank

Lab Goal

Before we get started building custom services with Scripted REST APIs, we need to get our lab instance setup. In this lab you will be modifying an existing scoped application. Start out by importing the Polls Application from Source Control. Follow the directions below to fork this application to your GitHub account and begin working.

Lab Setup

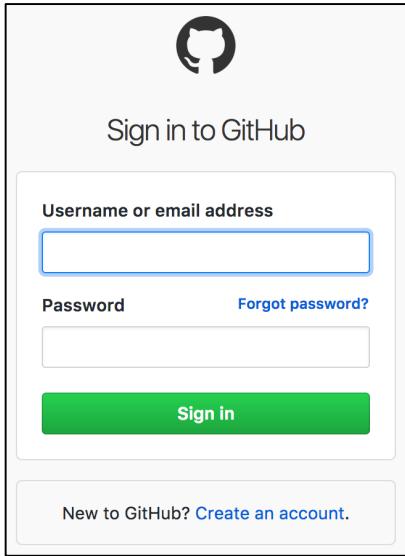
Prerequisites

In order to complete this lab, you must:

- Create a GitHub account, if you do not already have one.
- Install Postman from <https://getpostman.com> if you do not already have it.

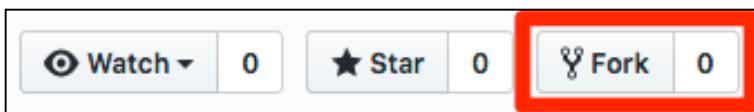
Fork the Lab GitHub Repository

1. Log in to your GitHub account at <https://github.com/login>.



2. Navigate to: <https://github.com/balazsburgermeister/ScriptedRESTAPI>

3. Click **Fork**.

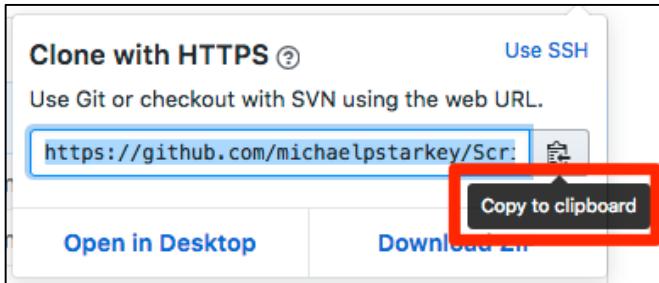


4. Note in the upper left that the repository has been copied to your account. You now have a copy of the lab material for reference after the conference!

5. Locate and click on the **Clone or download** button and then click the clipboard to the right.

This action copies the URL in the clipboard.

IMPORTANT: Be sure to copy the **HTTPS** repo URL in GitHub.

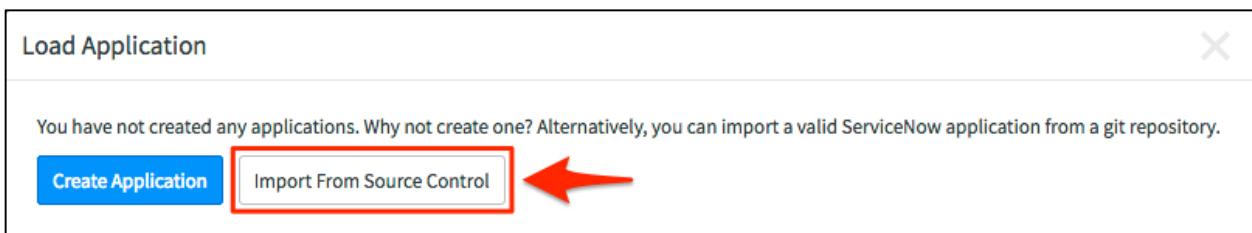


Import the Polls Application from Source Control

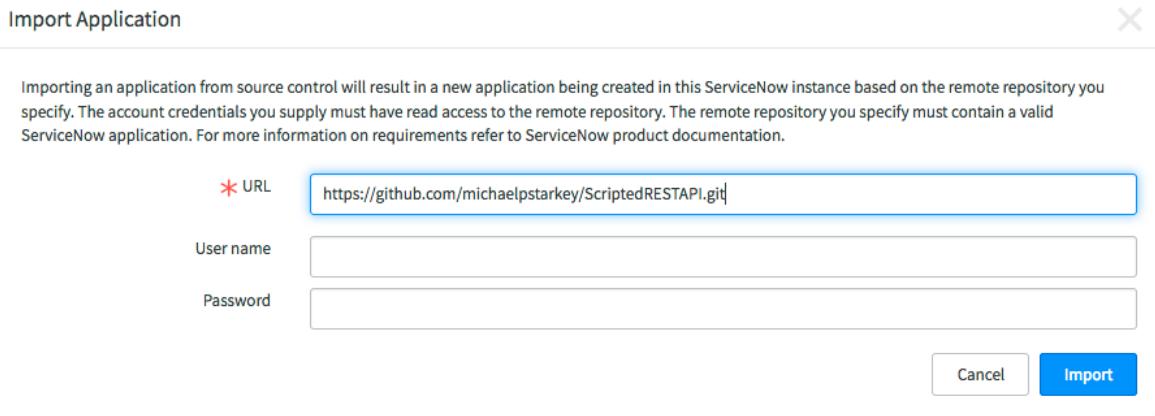
6. Log in to your instance with the credentials provided on the cover sheet of this document.
7. Navigate to **System Applications > Studio**.



8. Click **Import From Source Control**.

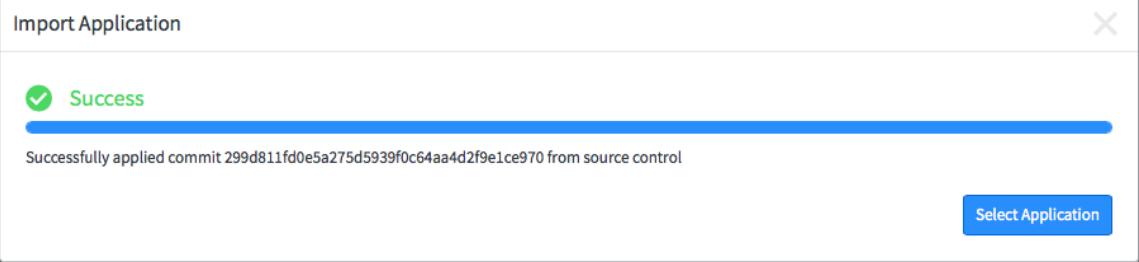


9. In the Import Application window, paste the URL copied in step 5 and provide your GitHub credentials. Click **Import**.



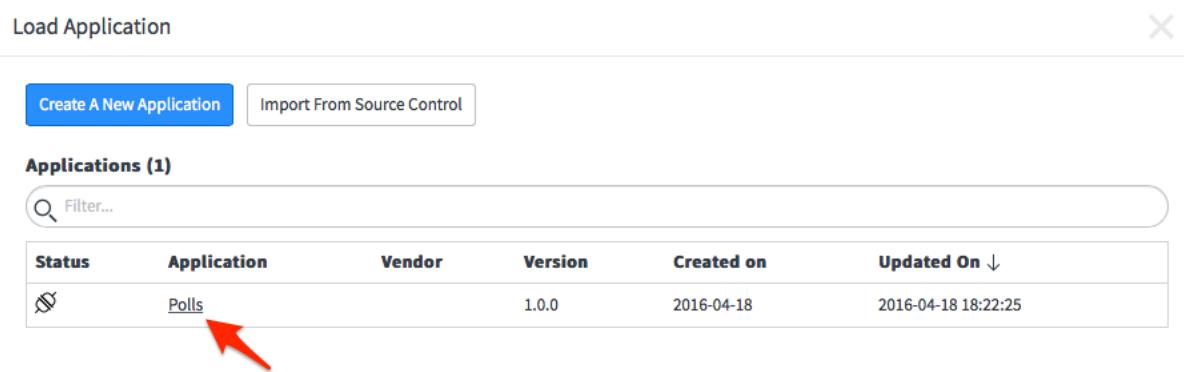
The screenshot shows the 'Import Application' dialog box. At the top, there is a note about importing from source control. Below it, there are three input fields: 'URL' containing 'https://github.com/michaelpstarkey/ScriptedRESTAPI.git', 'User name' (empty), and 'Password' (empty). At the bottom right are 'Cancel' and 'Import' buttons.

10. When the import completes, click **Select Application**.



The screenshot shows the 'Import Application' dialog box after a successful import. It displays a green checkmark icon and the word 'Success'. Below that, a message says 'Successfully applied commit 299d811fd0e5a275d5939f0c64aa4d2f9e1ce970 from source control'. At the bottom right is a blue 'Select Application' button.

11. Click on the **Polls** application you just imported.



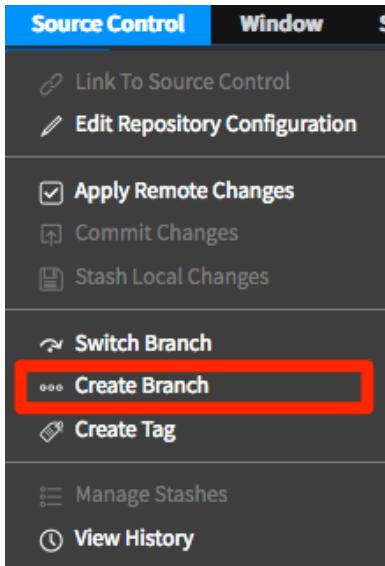
The screenshot shows the 'Load Application' dialog box. It has two buttons at the top: 'Create A New Application' (blue) and 'Import From Source Control' (white). Below that is a section titled 'Applications (1)' with a search bar. A red arrow points to the 'Polls' application in the list, which includes columns for Status, Application, Vendor, Version, Created on, and Updated On.

Status	Application	Vendor	Version	Created on	Updated On ↓
	Polls		1.0.0	2016-04-18	2016-04-18 18:22:25

You've now successfully imported your forked version of the application for use in this workshop.

Get ready for Lab 1 – Create a new branch from Lab1-start tag in Studio

12. In Studio, navigate to Source Control > Create Branch.



13. In the pop-up window, enter a branch name, then select **Lab1-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab1-branch**

Create from Tag: **Lab1-start**

The dialog box has a title 'Create Branch' and a close button 'X'. A message states: 'Creating a branch will result in a new branch being created in the remote repository that is configured for this application.' There are two input fields: 'Branch Name' with value 'my-Lab1-branch' and 'Create from Tag' with value 'Lab1-start'. Below the fields are 'Cancel' and 'Create Branch' buttons.

14. When the create is complete, click **Close Dialog** in the Create Branch pop-up.

15. Verify Studio is on branch **my-Lab1-branch** from the bottom right corner of the screen.



Lab setup is complete. You are now ready to start Lab 1.

Lab Goal

The purpose of this lab is to familiarize yourself with ServiceNow Scripted REST APIs. In this first lab you'll build a Scripted REST API that returns "Hello, world!" in response to a GET request. After building the API you'll use the ServiceNow REST API Explorer and API testing tool Postman to make requests to the REST API.

Prerequisite

- Knowledge of REST APIs
- Knowledge of HTTP clients
- Postman API testing tool. To get Postman go to: <https://www.getpostman.com/>

Lab 1
Build
“Hello,
world!”

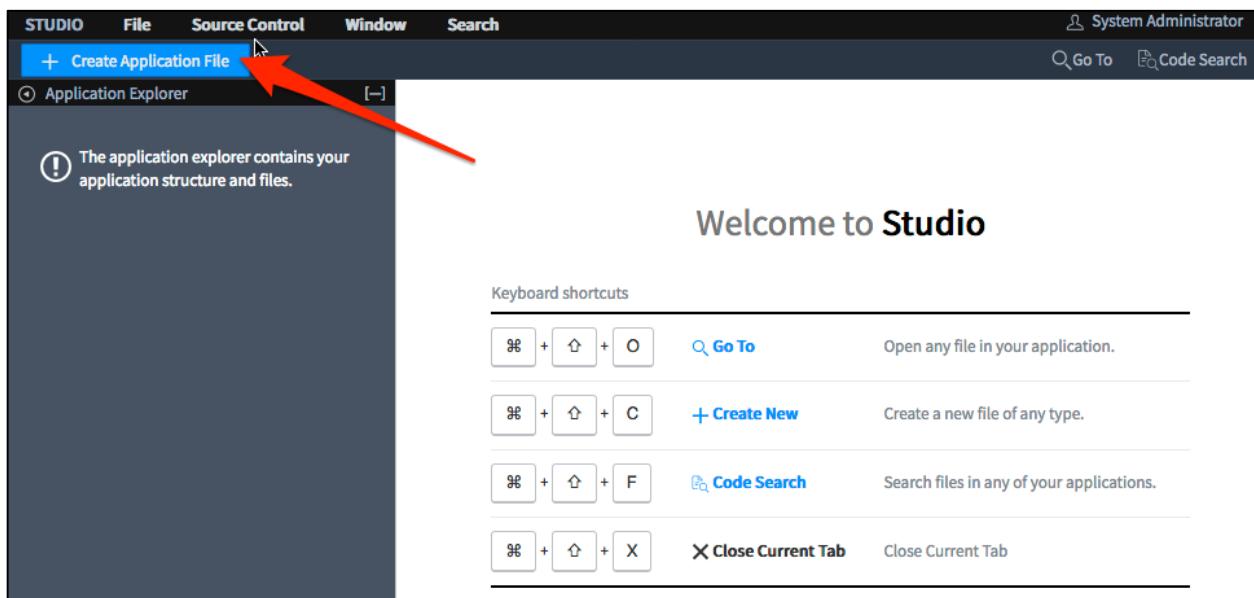
Create Lab 1 starting branch

1. If you completed the lab setup, proceed to the next step.

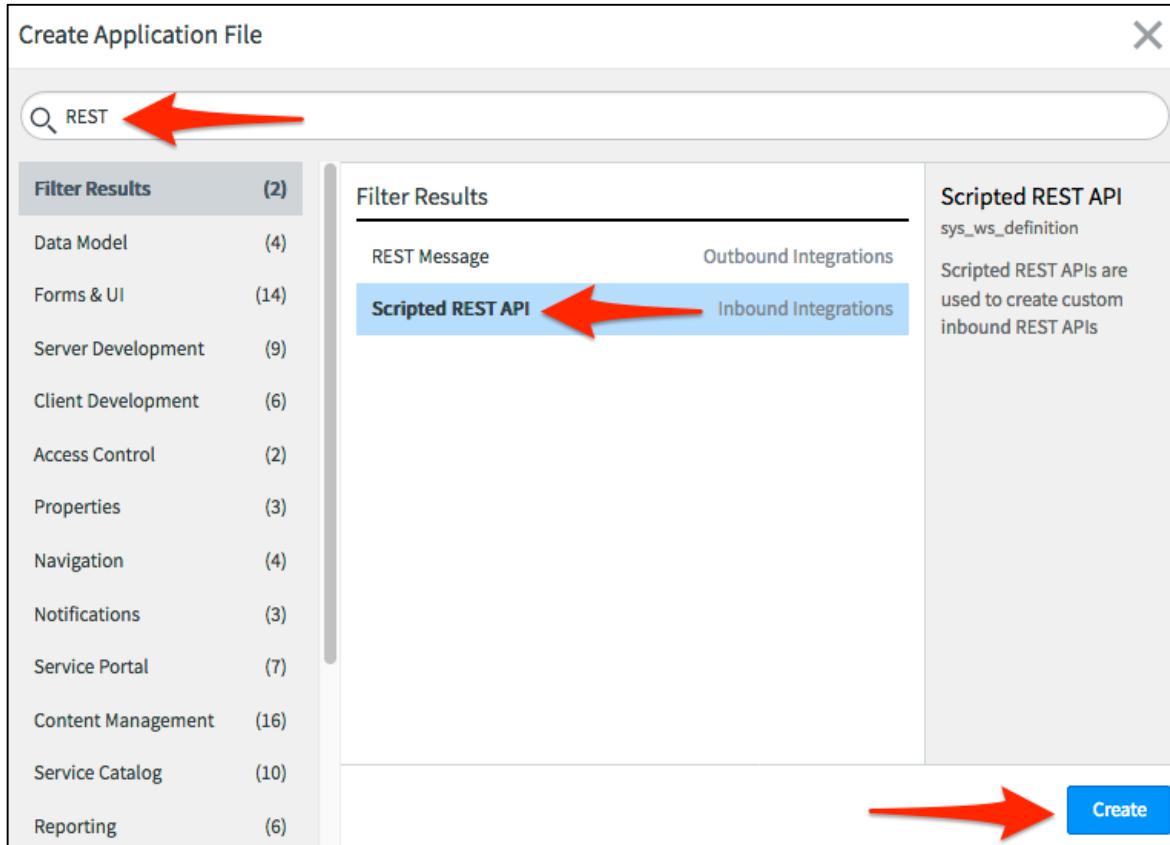
If you haven't yet completed lab setup, follow the steps in lab setup to create the **my-Lab1-branch** from the **Lab1-start** git tag.

Create the Hello World Scripted REST API

2. In Studio, click **Create Application File**.



3. In the **Create Application File** window, type **REST** in the filter then select **Scripted REST API** and click **Create**.



4. Give the **API** a name. Note the **API ID** populates automatically from the API Name, but can be changed.

Name: **Hello, world!**

Hello, world!
Scripted REST API

Scripted REST Service
New record

You can easily create a new REST API. To get started, give your API a name and ID. X

* Name: Hello, world!

* API ID: hello_world

Application: Polls (i)

* API namespace: x_snc_polls

Protection policy: -- None --

Submit

Click **Submit**.

5. Add a **Resource** to the API by finding the **Resources** related list. Click **New**.

Related Links:
[Enable versioning](#)
[Explore REST API](#)
[API analytics](#)

Resources Request Headers Query Parameters

Resources New

API definition = Hello, world!

(i) Search

Name ▲ HTTP method Relative path Resource path API version

No records to display

6. Specify the following properties for the new resource and complete the script.

Name: **Hello resource**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab1

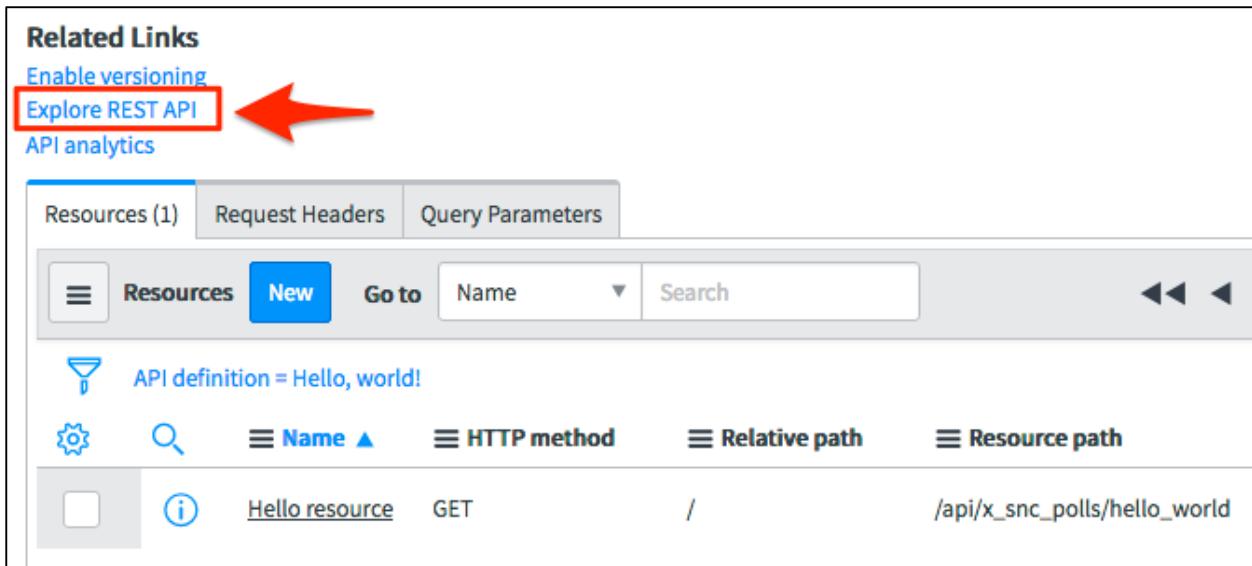
The screenshot shows the 'Scripted REST Resource' configuration page for a resource named 'Hello resource'. The 'Name' field is highlighted with a red box and an arrow pointing to it. The 'Script' section contains the following code:

```
1 v (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2   return "Hello, world!";  
3 })(request, response);
```

Click **Submit**.

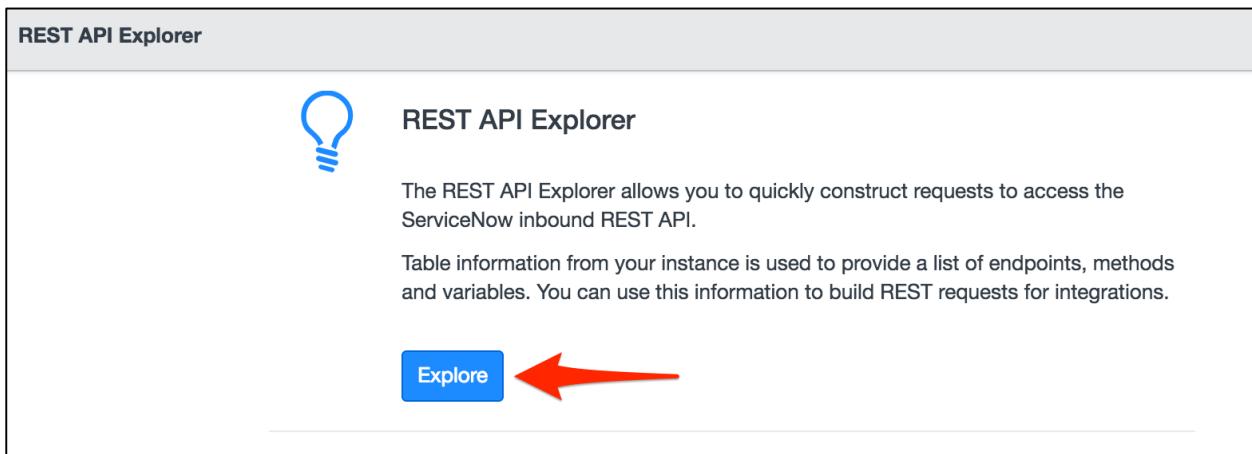
Test with REST API Explorer

7. Click **Explore REST API**.



The screenshot shows the ServiceNow navigation bar with several links: 'Enable versioning', 'Explore REST API' (which has a red arrow pointing to it), and 'API analytics'. Below the bar is a search bar with fields for 'Resources (1)', 'Request Headers', and 'Query Parameters'. Underneath the search bar is a table header with columns: 'Resources', 'New', 'Go to', 'Name', 'Search', and three navigation arrows. The table body contains one row: 'Hello resource' (with a gear icon) and 'GET' (with a magnifying glass icon). To the right of the table are the relative path '/api/x_snc_polls/hello_world' and the resource path 'Hello resource'.

8. The **REST API Explorer** opens in a new browser window. Click **Explore**.



The screenshot shows the REST API Explorer page. At the top left is the title 'REST API Explorer' with a lightbulb icon. The main content area contains a brief description: 'The REST API Explorer allows you to quickly construct requests to access the ServiceNow inbound REST API.' Below this is another paragraph: 'Table information from your instance is used to provide a list of endpoints, methods and variables. You can use this information to build REST requests for integrations.' At the bottom of the page is a blue 'Explore' button, which has a red arrow pointing to it.

9. The “Hello, world!” Scripted REST API is pre-selected in the Explorer menus. Click **Send**.

The screenshot shows the REST API Explorer interface. On the left, there are three dropdown menus: 'Namespace' (set to 'x_snc_polls'), 'API Name' (set to 'Hello, world!'), and 'API Version' (set to 'latest'). To the right of these is a large red box containing the API details: 'Hello, world!', 'Hello resource', and a 'GET /api/x_snc_polls/hello_world' endpoint. Below this is a 'Prepare request' section with 'Query parameters' and 'Request headers' tables. The 'Request headers' table has three rows: 'Request format' (Value: application/json), 'Response format' (Value: application/json), and 'Authorization' (Value: Send as me). At the bottom of the interface is a blue 'Send' button, which is also highlighted with a red arrow.

10. Verify response status code is **200 OK** and response body is “Hello, world!”.

Request	
HTTP Method / URI	GET /api/x_snc_polls/hello_world
Headers	
Accept	application/json
X-UserToken	cbc451c1f51232007f4494386d91c994797a9010e7105820322bb160a4a94b657e7dcd0c
Response	
Status code	200 OK 
Headers	
Cache-control	no-cache,no-store,must-revalidate,max-age=-1
Content-encoding	gzip
Content-type	application/json;charset=UTF-8
Date	Fri, 21 Apr 2017 11:45:21 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-encoding	chunked
X-is-logged-in	true
Response Body	
{ "result": "Hello, world!" }	

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

11. Similar to creating the Lab1 starting branch, the completed lab can also be checked out from a tag (**Lab1-complete**) in Source control.

12. In **Studio**, navigate to **Source Control > Create Branch**.

13. In the pop-up window, enter a branch name, then select **Lab1-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab1-branch-complete**

Create from Tag: **Lab1-complete**

14. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

15. Verify Studio is on branch **my-Lab1-branch-complete**.

16. You are now ready to continue with the next section of **Lab 1**.

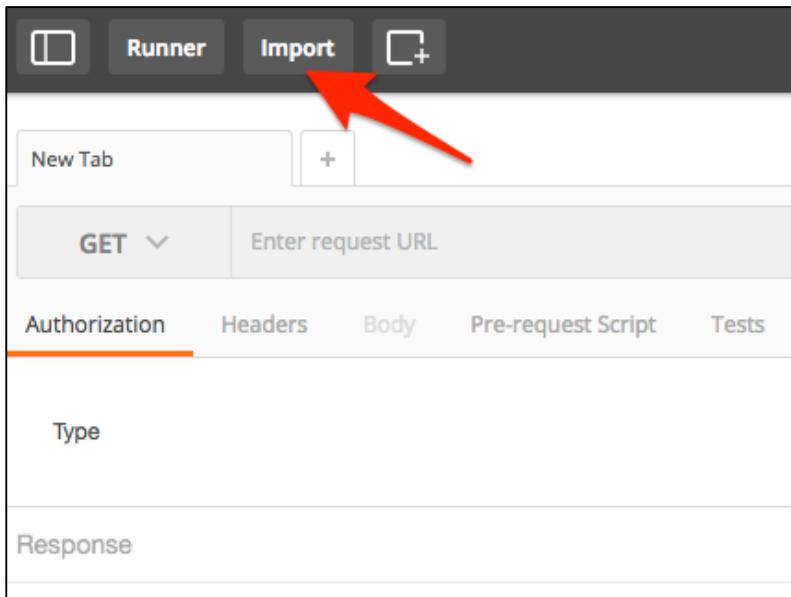
Test with Postman

17. Open the **Postman** application on your laptop.

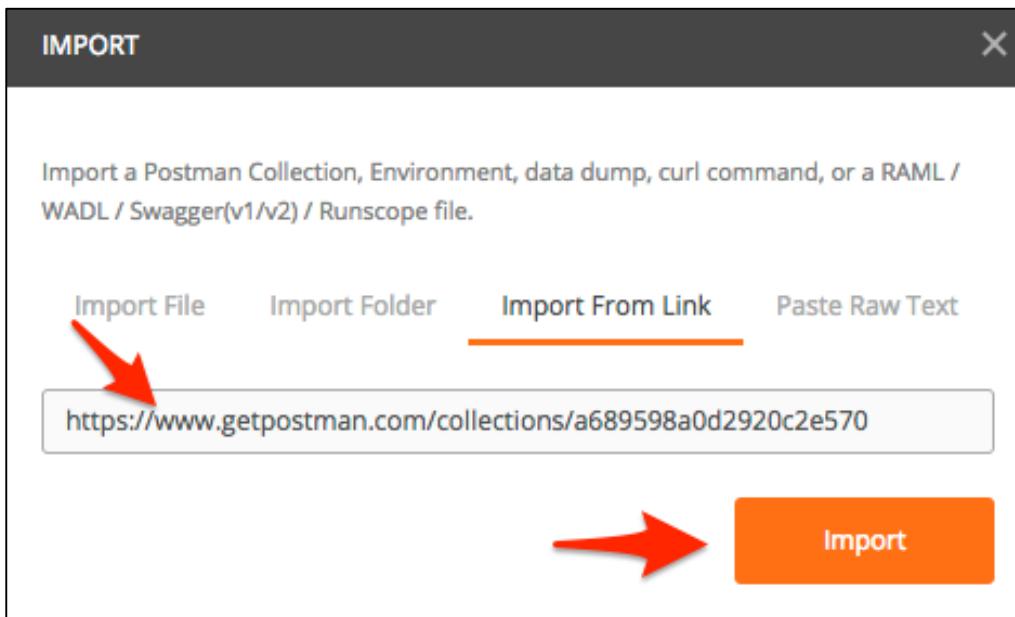
18. Import the Postman collection we will be using for this workshop from:

Postman Collection Link: <https://www.getpostman.com/collections/a689598a0d2920c2e570>

19. In Postman, click **Import**.



20. Paste the link to our Postman collection in the **Import from Link** and click **Import**.

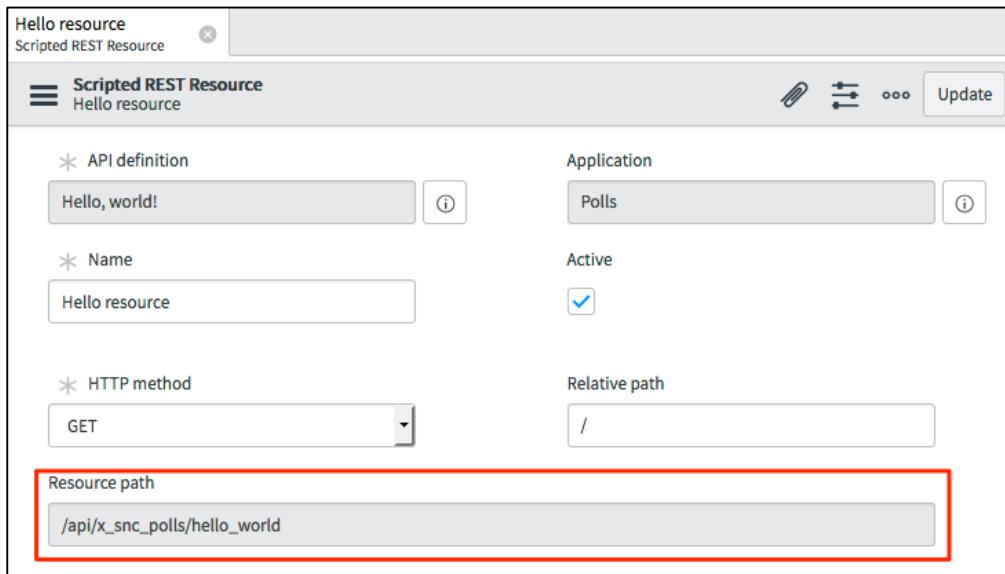


21. Verify you have the “Scripted REST API Workshop” collection loaded by searching for it in the navigator on the left hand side.

The screenshot shows the Postman application interface. At the top, there are tabs for 'Runner', 'Import', and a new one. The 'Builder' tab is active. In the top navigation bar, there is a search bar with the text 'SCRIPTED REST API WORKSHOP' and a red arrow pointing to it. Below the search bar, there are tabs for 'History' and 'Collections'. The 'Collections' tab is selected, showing a folder named 'SCRIPTED REST API WORKSHOP' which contains 19 requests. A red arrow points to this folder. On the right side, the 'Builder' panel is open, showing a request for 'CC17: Intro Lab'. The request method is 'GET', the URL is '{{instance_url}}/api/x_snc_polls/hellow...', and the 'Headers (1)' tab is selected. Under 'Headers', there is a key 'Authorization' with a value '{{basic_auth_header}}'. There are also tabs for 'Params', 'Send', 'Save', and 'Code'.

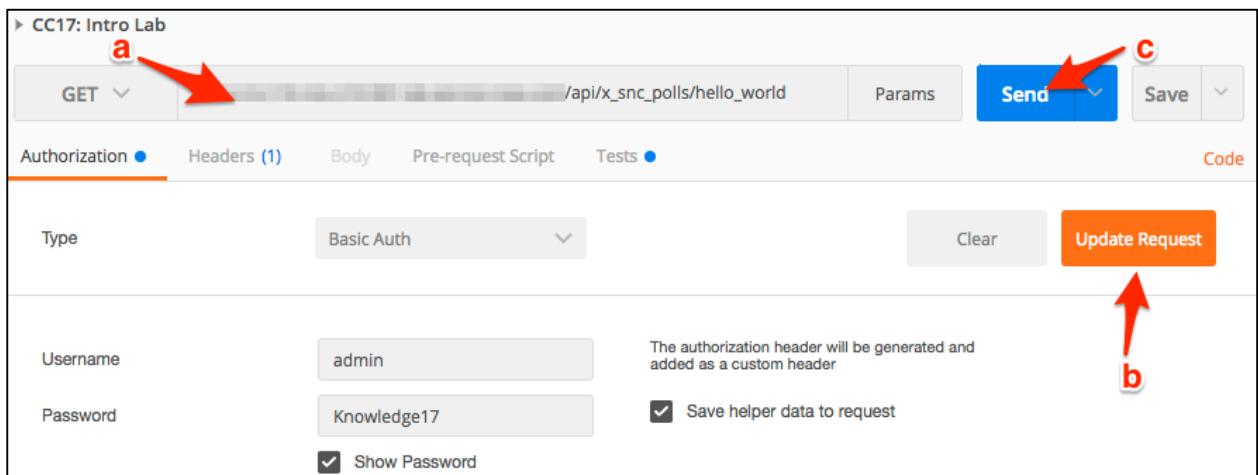
22. In the **Scripted REST API Workshop** select the **CC17: Intro Lab**.

- a. Replace {{instance_url}} with your lab instance URL (for example, https://my_instance.lab.service-now.com), and replace the resource URI with the resource from your Hello World Scripted REST API. Copy/paste the resource path from the **Resource path** field.



The screenshot shows the 'Hello resource' configuration in the Scripted REST Resource section. The 'Resource path' field contains '/api/x_snc_polls/hello_world' and is highlighted with a red border. Other fields include 'API definition' (Hello, world!), 'Name' (Hello resource), 'HTTP method' (GET), 'Relative path' (/), and 'Active' status (checked).

- b. Click **Update Request**.
- c. Click **Send** to send the HTTP request.



The screenshot shows a POSTMAN request configuration for the 'CC17: Intro Lab'. The 'Authorization' tab is selected (labeled 'a'). The request method is 'GET' (labeled 'a'). The URL is '/api/x_snc_polls/hello_world'. The 'Send' button is highlighted with a red arrow (labeled 'c'). The 'Update Request' button is also highlighted with a red arrow (labeled 'b'). Other tabs include 'Headers (1)', 'Body', 'Pre-request Script', 'Tests', and 'Code'. Authorization details show 'Type' as 'Basic Auth', 'Username' as 'admin', and 'Password' as 'Knowledge17'. A note says 'The authorization header will be generated and added as a custom header'. There is also a checked checkbox for 'Save helper data to request'.

23. Validate response is successful by looking for the **200 OK** status code and message and that the response payload contains “Hello, world!”.

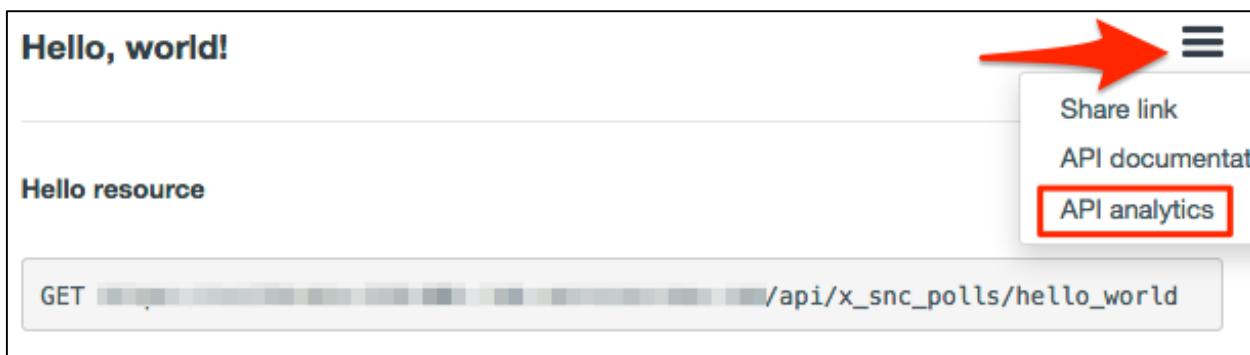
The screenshot shows the Postman application interface. At the top, there's a header bar with 'CC17: Intro Lab' and several buttons: 'Send', 'Save', 'Code', 'Params', and 'Headers (1)'. Below this is a toolbar with 'GET', a URL field containing '/api/x_snc_polls/hello_world', and dropdown menus for 'Type' (set to 'Basic Auth'), 'Authorization' (selected), 'Headers (1)', 'Body', 'Pre-request Script', and 'Tests (1/4)'. On the right side of the toolbar are 'Clear' and 'Update Request' buttons. In the main body area, under 'Authorization', 'Username' is set to 'admin' and 'Password' is set to 'Knowledge17'. There's a note: 'The authorization header will be generated and added as a custom header'. A checked checkbox says 'Save helper data to request'. Below this is a 'Body' tab with sub-options 'Pretty', 'Raw', and 'Preview'. The 'Pretty' option is selected, showing a JSON response:

```
1 {  
2   "result": "Hello, world!"  
3 }
```

A red arrow points to the 'Pretty' tab. Another red arrow points to the JSON response itself. To the right of the response, it says 'Status: 200 OK' and 'Time: 1332 ms'. At the bottom right of the main area are 'Save Response' and search icons.

View API Analytics for Hello World

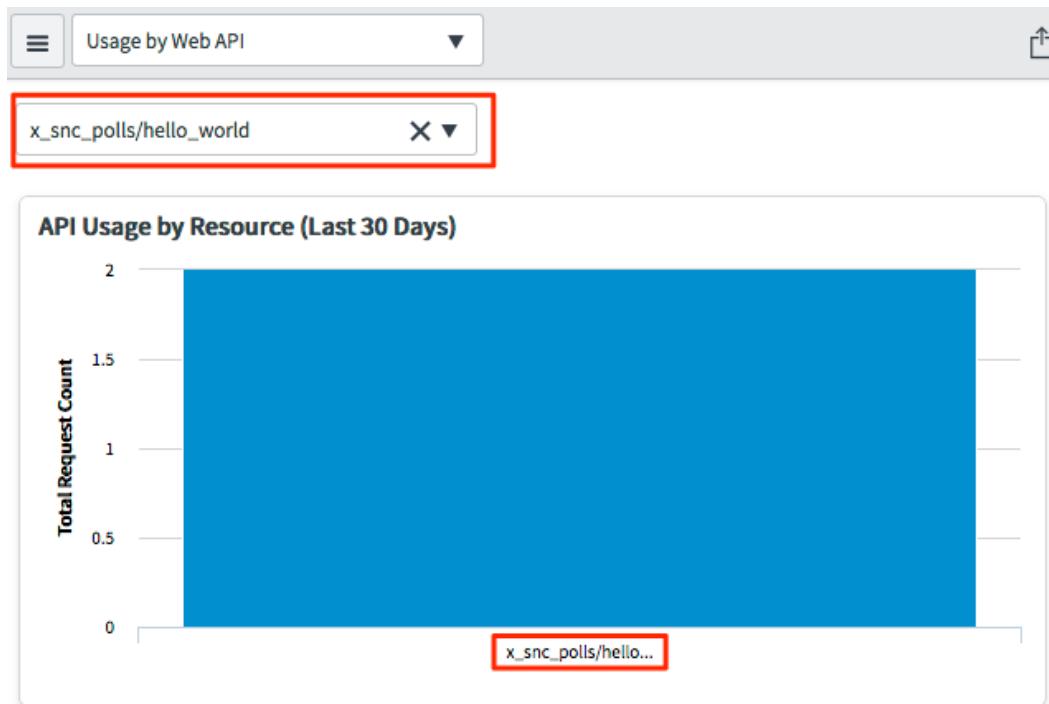
24. From **Explorer context menu**, or from Scripted REST API definition click **API Analytics**.



or

A screenshot of the ServiceNow API Explorer interface. At the top left, there is a "Related Links" section with links to "Enable versioning", "Explore REST API", and "API analytics". The "API analytics" link is highlighted with a red box and has a red arrow pointing to it. Below this is a search bar with tabs for "Resources (1)", "Request Headers", and "Query Parameters". The "Resources" tab is selected. Underneath is a table with columns: "Name", "HTTP method", "Relative path", and "Resource path". The table shows one entry: "Hello resource" (Name), "GET" (HTTP method), "/" (Relative path), and "/api/x_snc_polls/hello_world" (Resource path). There are also filter and search icons at the top of the table.

25. The **API Analytics** usage dashboard opens in a new browser tab, with the **Hello world API** pre-selected. Observe the API counts.



Note: There is up to a 60s delay between an API call and when it is reflected in API Analytics.

26. Close the REST API Explorer and API Analytics dashboard windows.

Lab 1 is complete. You are now ready to start lab 2.

Lab Goal

Having familiarized yourself with Scripted REST APIs in Lab 1, in Lab 2 we'll start building the "Polls" REST API that we'll use for the rest of this workshop. The Polls API you'll build provides a programmatic interface to interact with the Polls application on your ServiceNow instance.

The Polls app is a simple app that allows for the creation of Polls that allow participants to vote on answers to questions. Polls can have one or more questions associated with them. Questions can have one or more choices associated with them. As an example a simple poll could contain the question "What is your favorite color?". Choices that participants could choose would be; blue, red, yellow.

Lab 2 Building the Polls REST API

Create Lab 2 starting branch

1. In **Studio**, navigate to **Source Control > Create Branch**.
2. In the pop-up window, enter a branch name, then select **Lab2-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab2-branch**
Create from Tag: **Lab2-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab2-branch**.
5. You are now ready to start **Lab 2**.

Create the Polls Scripted REST API

6. In Studio, click **Create New Application File**.
7. In the **Create New Application File** window, type **REST** in the filter then select **Scripted REST API** and click **Create**.
8. Give the **Scripted REST API** a name, then click **Submit**.

Name: **Polls**

API ID: **polls**

9. Click the related link **Enable versioning** to enable versioned URIs for the new API.

BEST PRACTICES

Do: Use versioning to control API changes.

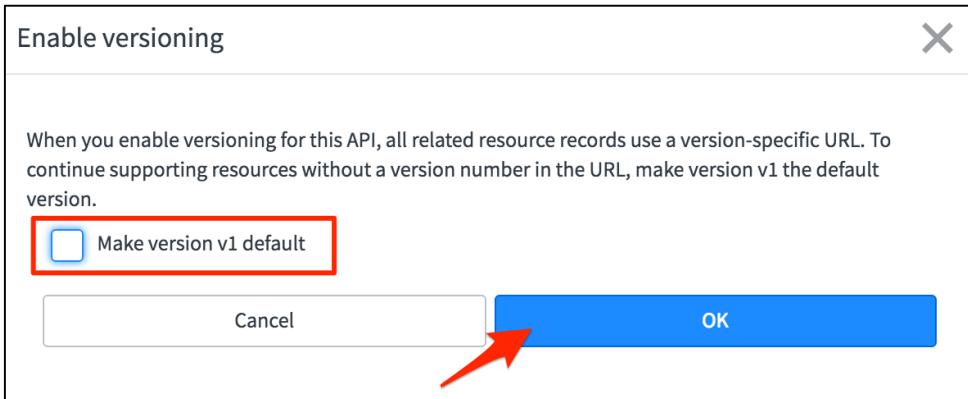
Do: Encourage clients to integrate against specific versions.

Don't: Make breaking changes in an existing version.

Do: Release a new API version when introducing new behaviors.

The screenshot shows the 'Scripted REST Service' interface for creating a new application file named 'Polls'. The form includes fields for Name (Polls), API ID (polls), and Active status (checked). A 'Protection policy' dropdown is set to '-- None --'. Below the form are tabs for Security, Content Negotiation, and Documentation, with 'Security' currently selected. Under 'Default ACLs', there is a lock icon. At the bottom are 'Update' and 'Delete' buttons. A 'Related Links' section contains three items: 'Enable versioning' (which has a red arrow pointing to it), 'Explore REST API', and 'API analytics'.

10. In the Enable versioning popup, uncheck the **Make version v1 default** checkbox, then click **OK**.



11. A new tab **Versioning** appears. Click to review the versioning tab contents.

Version ID	Is default	Active	Deprecated
v1	false	true	false

Note: The API versions are maintained here. Deactivate versions, mark a version **Is default=true** to allow non-versioned URIs to route to that version, or don't define a default version to force clients to specify the version when making requests to the API.

12. Add a **Resource** to the API. Click **New** on the **Resources** related list.

This resource will return the details of a specific poll.

13. Specify the following properties for the new resource and complete the script.

Name: **Retrieve poll detail**

API Version: **v1**

HTTP method: **GET**

Relative path: **{poll_id}**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab2_retrieve_poll_details

The screenshot shows the 'Scripted REST Resource' configuration page for a resource named 'Retrieve poll details'. Key fields highlighted with red boxes are: 'Name' (containing 'Retrieve poll details'), 'API version' (containing 'v1'), 'HTTP method' (containing 'GET'), and 'Relative path' (containing '{poll_id}'). The 'Request routing' section describes the route configuration. The 'Implement the resource' section contains a script editor with the following code:

```
1 v  (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2   var id = request.pathParams.poll_id;  
3   //...  
4 } )
```

Click **Submit**.

Test with REST API Explorer

14. Click **Explore REST API**,

The screenshot shows the REST API Explorer interface. In the top left, there's a 'Related Links' section with 'Add new version', 'Explore REST API' (which is highlighted with a red box), and 'API analytics'. Below that is a navigation bar with tabs: 'Resources (1)', 'Request Headers', and 'Query Parameters'. The main area displays a table of resources. A filter at the top says 'API definition = Polls'. The table has columns: Name, HTTP method, Relative path, Resource path, API version, and Active. One row is selected: 'Retrieve poll detail' (Name), GET (HTTP method), '/{poll_id}' (Relative path), '/api/x_snc_polls/v1/polls/{poll_id}' (Resource path), 'v1' (API version), and 'true' (Active). At the bottom, there are buttons for 'Actions on selected rows...' and navigation controls.

15. The “**Polls**” Scripted REST API is pre-selected in the Explorer menus and the **Retrieve Poll detail** resource is preselected.

16. Fill in sys id for a demo poll record and make a request.

To get the sys_id of demo record. Open the Polls module from navigator. Right click on existing record to copy sys_id.

The screenshot shows the ServiceNow Polls module. On the left, there's a sidebar with a 'polls' search input field. The main area shows a table of 'Poll' records. One record, 'First Poll' (Number: POL0001002), is selected and has a context menu open. The menu items are: 'Right click to show context menu', 'Show Matching', 'Filter Out', 'Copy URL to Clipboard', and 'Copy sys_id' (which is highlighted with a red arrow). Navigation controls are at the bottom right.

17. Fill in the sys_id on the REST API Explorer.

The screenshot shows the REST API Explorer interface. On the left, there are dropdown menus for Namespace (x_snc_polls), API Name (Poll), and API Version (v1), all of which are highlighted with a red box. Below these is a button labeled "Retrieve poll details (GET)" also highlighted with a red box. To the right, under the heading "Poll", is a description: "Retrieve poll details - Retrieve poll, questions, choices and votes polled by sysid". Below this is a "Prepare request" section with a "Path parameters" table. The table has two rows: one for "Name" (poll_id) and one for "Value" (ddee64b9443a1200964fac543127a1ab), both of which are highlighted with a red box. An arrow points from the "Value" field up towards the "Path parameters" header.

Click **Send**.

18. Verify the response status code is **200-OK**.

The screenshot shows the REST API Explorer response details. At the top, it says "Response". Under "Status code", it shows "200 OK" in green. Below that is a table of "Headers" with various key-value pairs. Under "Response Body", it shows a JSON object:

```
{  
  "result": {  
    "name": "First poll",  
    "questions": [  
      {  
        "id": "2a46dae6134b1200ed373d62f244b041",  
        "question": "Favorite number",  
        "choices": [  
          {  
            "id": "3aa61ee6134b1200ed373d62f244b0dc",  
            "choice": "3",  
            "score": null  
          }  
        ]  
      }  
    ]  
  }  
}
```

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

19. Similar to creating the Lab2 starting branch, the completed lab can also be checked out from a tag (**Lab2-complete**) in Source control.

20. In **Studio**, navigate to **Source Control > Create Branch**.

21. In the pop-up window, enter a branch name, then select **Lab2-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab2-branch-complete**

Create from Tag: **Lab2-complete**

22. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

23. Verify Studio is on branch **my-Lab2-branch-complete**.

24. You are now ready to continue with the next section of **Lab 2**.

Test with Postman

So far in this lab you've used Postman to make requests to ServiceNow REST APIs, Postman also allows you to write and execute tests that evaluate response from a REST API and provide you with Pass/Fail information based on your test and the response from the request. Let's issue a request against the new resource **Retrieve poll detail** and write a few tests to verify the response we receive.

25. In Postman select the **CC17: Retrieve poll detail** request in the Scripted REST API Workshop collection. This request has been pre-built for you however you will need to update the {{instance_url}} and path including {{poll_id}} parameters in the URL replacing them with values from your lab instance. You will also need to update the Authorization section specifying your username and password.

- instance_url: **URL of your lab instance**
- poll_id: **Sys_id of a poll record in your lab instance**
- Username: admin
- Password: admin password for your lab instance

▶ CC17: Retrieve poll detail

a {{instance_url}}/api/x_snc_polls/v1/poll/{{poll_id}}

b

GET ▼ Headers (1) Body Pre-request Script Tests ● Save ▾ Code

Type Basic Auth ▾ Clear Update Request

Username admin The authorization header will be generated and added as a custom header

Password Knowledge17 Save helper data to request

Show Password

a. Replace {{instance_url}} with the URL of your lab instance (e.g., https://mylabinstance.service-now.com).

b. Replace {{poll_id}} with the sys_id of an existing Poll record in your ServiceNow instance.

26. After populating your credentials and replacing the parameters click **Update Request** and then **Send** the request.

27. Check that you've received a successful response. You should see a status of **200 OK** and a JSON payload that includes at least one poll as shown below.

▶ CC17: Retrieve poll detail

GET ▼ /api/x_snc_polls/v1/poll/3c265ae6134b... Params Send ▾ Save ▾ Code

Authorization ● Headers (1) Body Pre-request Script Tests ●

Type Basic Auth ▾ Clear Update Request

Username admin The authorization header will be generated and added as a custom header

Password Save helper data to request

Show Password

Body Cookies Headers (10) Tests (3/3) Status: 200 OK Time: 240 ms

Pretty Raw Preview JSON ▾

1 ~ {
2 ~ "result": {
3 ~ "name": "First poll",
4 ~ "questions": [
5 ~ {
6 ~ "id": "2a46dae6134b1200ed373d62f244b041",
7 ~ "question": "Favorite number",
8 ~ "choices": [
9 ~ {
10 ~ "id": "3aa61ee6134b1200ed373d62f244b0dc",
11 ~ "choice": "3",
12 ~ "score": null
13 ~ },
14 ~ {
15 ~ "id": "41a616e6134b1200ed373d62f244b0fc",
16 ~ }

28. This request should return a status code of **200 OK**, with a JSON payload that represents the poll we requested. In addition the content-type header in the response should be **application/json;charset=UTF-8** and our JSON payload should contain a **result** object. Let's see how we can use Postman to verify this for us with tests that will be run as part of the request.

29. In Postman, in the CC:17 Retrieve poll detail request, open the **Tests** tab by clicking on **Tests**. Here you can specify tests that will be run as part of each request.

The screenshot shows the Postman interface for a request titled "CC17: Retrieve poll detail". The request method is GET, and the URL is /api/x_snc_polls/v1/poll/3c265ae6134b... The "Tests" tab is selected, indicated by a red arrow. The test script contains three lines of code:

```
1 tests["Status code is 200"] = responseCode.code === 200;
2 tests["Content-Type is application/json;charset=UTF-8"] = postman.getResponseHeader
  ("Content-Type") === "application/json;charset=UTF-8";
3 tests["Response Body Contains result"] = responseBody.has("result");
```

A red arrow points to the third line of the test script. To the right of the main window is a sidebar titled "SNIPPETS" containing various test snippets.

30. Postman has its own simple syntax for declaring tests. You can find out more about this syntax at the Postman website. For this lab we've provided you with 3 tests that validate that:

- the response status code is **200**
- the response includes a content-type header with a value of **application/json;charset=UTF-8**
- the response body contains the text '**result**'

31. Now update your request in Postman to run these tests. Copy the test script from the following URL and paste it into the **Tests** area in Postman.

- Postman test script: http://bit.ly/CC17_ScriptedRESTAPI_Lab2_postman_test_script

32. After copying click **Send** to issue the request. Now that we have tests specified as part of our request test results will be displayed in the response area. If all tests passed you will see a '(3/3)' in the header and then a green **PASS** image next to each test as shown below.

The screenshot shows the Postman interface after a request has been sent. At the top, there are tabs for Body, Cookies (3), Headers (9), and Tests (3/3). A red arrow points to the 'Tests (3/3)' tab. Below it, the status bar shows 'Status: 200 OK' and 'Time: 118 ms'. The main area displays three test results, each with a green 'PASS' icon and a brief description: 'Status code is 200', 'Content-Type is application/json;charset=UTF-8', and 'Response Body Contains result'. A red arrow also points to the first test result.

Save your request in Postman. You now have a saved request in Postman that allows you to easily issue a request to your 'Retrieve poll detail' resource and which will run test the response to validate that it includes the correct status code, header, and payload content. These were simple test cases but Postman will allow you to define more advanced test cases to verify you are receiving the correct response from your REST API.

BEST PRACTICES

Do: Define test cases for each of your APIs resources to validate that the response is formatted correctly and that the response contains the intended content. Building test cases as part of your development process will help insure you're building the API as you designed it and provide you with a set of tests that can be run over time as you make changes to guarantee that your interface has not changed unintentionally.

33. Close the REST API Explorer and API Analytics dashboard windows.

Lab 2 is complete. You are now ready to begin lab 3.

Lab Goal

In Lab 3 you'll continue building out the REST API for the **Polls** application adding resources to support creating a new poll, editing an existing poll, and voting in a poll. In building out this additional functionality you will further use and familiarize yourself with the Request and Response APIs that allow you to interact with the request that your REST API receives and build the response that your REST API will return.

Lab 3 Request & Response API

Create the Lab 3 starting branch

1. In Studio, navigate to Source Control > Create Branch.
2. In the pop-up window, enter a branch name, then select Lab3-start from the Create from Tag menu, and click **Create Branch**.

Branch: **my-Lab3-branch**
Create from Tag: **Lab3-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab3-branch**.
5. You are now ready to start **Lab 3**.

Create New Resource in Polls API - Create a poll

6. The ‘Create a poll’ resource will be used to create a new poll in the ‘Polls’ application. Open the Polls API in studio and click **New** on the **Resources** related list to create a new resource.

The screenshot shows the ServiceNow API Studio interface. The left sidebar has a 'Create New Application File' button highlighted in blue. The main area shows a 'Poll' resource under a 'Scripted REST API'. A red arrow points from the 'Poll' entry in the 'Scripted REST Resources' list to the 'New' button in the 'Resources' list below. The 'Resources' list table has the following data:

Name	HTTP method	Relative path	Resource path	API version	Active
Retrieve poll details	GET	{poll_id}	/api/x_snc_polls/v1/poll/{poll_id}	v1	true

7. Specify the following properties for the new resource.

Name: **Create new poll**

API Version: **v1**

HTTP method: **POST**

Relative path: **/**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab3_create_new_poll

The screenshot shows the 'Create new poll' configuration page in the ServiceNow interface. The top section is titled 'Scripted REST Resource' with fields for 'Name' (highlighted with a red box) and 'API version' (also highlighted with a red box). Below this is a 'Request routing' section with 'HTTP method' set to 'POST' and 'Relative path' set to '/'. The 'Implement the resource' section contains a 'Script' tab where a JavaScript function is pasted:

```
1 v (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     var data = request.body.data;  
3     var pollHelper = new x_snc_polls.PollData_Creator();  
4     var groupId = getUserGroupId(data.usergroup);  
5  
6     // Create Poll record  
7     var pollRecord = pollHelper.createPoll(data, groupId);  
8  
9     // Create poll question and options  
10    pollRecord.createPollQuestion(data);  
11    pollRecord.createPollOptions(data);  
12  
13    response.end();  
14 } );
```

Click **Submit**.

Test with REST API Explorer

8. Open 'Create new poll' resource and Click **Explore REST API**.

The screenshot shows the REST API Explorer interface for the 'Create new poll' resource. The top navigation bar includes 'Create new poll' (highlighted with a red box), 'Scripted REST Resource', and 'Scripted REST Resource' (with a 'Create new poll' link). The main content area displays security settings: 'Requires authentication' (checked) and 'Requires ACL authorization' (unchecked). Below this are 'Update' and 'Delete' buttons. A 'Related Links' section contains 'Explore REST API' (highlighted with a red box) and 'API analytics'. The bottom section shows 'Request Header Associations' and 'Query Parameter Associations' tabs, with the 'Request Header Associations' tab active. It features a search bar and a table with one row: 'API resource = Create new poll', 'API request header', 'Example value', and 'Is required'. A note at the bottom states 'No records to display'.

9. Create New poll resource is shown in the REST API Explorer. Fill in request body in the raw tab under Request body section.

The screenshot shows the REST API Explorer interface. On the left, there are dropdown menus for Namespace (x_snc_polls), API Name (Poll), and API Version (v1). Below these is a red box highlighting the 'Create new poll (POST)' button. To the right, the 'Poll' section is displayed with the title 'Create new poll'. A POST request URL is shown: `http://10.11.91.87:16001/api/x_snc_polls/v1/poll`. Under the 'Prepare request' section, there are tabs for 'Query parameters' and 'Request headers'. The 'Request headers' table includes columns for Name, Value, and Description. It lists three entries: 'Request format' set to 'application/json', 'Response format' set to 'application/json', and 'Authorization' set to 'Send as me'.

A sample request payload can be found at:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_create_new_poll_sample_request

10. Copy the sample payload into the 'Raw' tab.

The screenshot shows the 'Request Body' tab in the REST API Explorer. At the top, there are two tabs: 'Builder' and 'Raw', with a red arrow pointing to the 'Raw' tab. Below the tabs is a text area containing a JSON payload. This payload defines a poll with a user group ('Software') and four questions. Each question has multiple choices: 'Audi', 'BMW', 'Corvette', and 'Tesla'. The JSON is enclosed in a red box. At the bottom of the tab, there are 'Send' and 'Clear response' buttons, and a footer with links to various scripting languages: ServiceNow Script, cURL, Python, Ruby, JavaScript, Perl, and Powershell.

```

{
  "usergroup": "Software",
  "questions": [
    {
      "question": "what is your favorite car",
      "choices": [
        {
          "choice": "Audi"
        },
        {
          "choice": "BMW"
        },
        {
          "choice": "Corvette"
        },
        {
          "choice": "Tesla"
        }
      ]
    }
  ]
}
  
```

[ServiceNow Script] [cURL] [Python] [Ruby] [JavaScript] [Perl] [Powershell]

Click **Send**.

11. Verify response status code is **201 Created**

Response	
Status code	201 Created
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 15:55:46 GMT
Expires	0
Location	http://10.11.91.87:16001/api/x_snc_polls/v1/poll/83e342dab1321200964f2e4c16efa08f
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "result": { "number": "POL0001001", "name": "Second poll" } }	

Create tests in Postman

12. In Postman select the ‘CC17: Create new poll’ request’. This is a pre-built request that already contains an appropriately formatted payload for the ‘Create new poll’ resource.

The screenshot shows the Postman interface with a sidebar on the left. The sidebar has a folder icon and the text 'SCRIPTED REST API WORKSHOP' followed by '19 requests'. Below this, a list of requests is shown with their method, name, and a small preview. The 'POST CC17: Create new poll' request is highlighted with a gray background, indicating it is selected. Other requests visible include 'GET CC17: Intro Lab', 'GET CC17: Retrieve poll detail', 'GET CC17: Retrieve poll detail w/ TEST', 'PATCH CC17: Edit poll', 'PATCH CC17: Edit poll w/ TEST', 'POST CC17: Create new poll' (highlighted), 'POST CC17: Create new poll w/ TEST', 'DEL CC17: Delete a poll', 'DEL CC17: Delete a poll w/ TEST', and 'POST CC17: Vote in poll'. There is also a '...' button at the bottom right of the list.

13. Update the request replacing the {{instance_url}} and authorization credentials appropriate for your lab instance. Use your admin credentials for this request. Once you’ve updated those values save and then send the request.

14. As you saw when you tested with the REST API Explorer a successful response will include a **201** status code, a JSON payload that includes the number for the newly created poll, and the response headers include a ‘Location’ header that provides the URL for this newly created record. Let’s add tests in Postman that verify that the following details for in the response:

- Response status code is 201
- Response headers include Location
- Response headers include Content-Type of application/json;charset=UTF-8
- Response body contains the text ‘number’
- Response body contains the text ‘name’

Update the request in Postman to include the following:

```
tests["Status code is 201"] = responseCode.code === 201;
tests["Location Header is present"] = postman.getResponseHeader("Location");
tests["Response Body Contains number"] = responseBody.has("number");
tests["Response Body Contains name"] = responseBody.has("name");
tests["Content-Type is application/json;charset=UTF-8"] = postman.getResponseHeader("Content-Type") === "application/json;charset=UTF-8";
```

For ease you can also copy these from:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_create_new_poll_test_script

15. After adding the tests above save your request in Postman and Send the request. In the response you should see the following tests and results.

The screenshot shows the Postman interface for a POST request to `https://[REDACTED].service-now.com/api/x_snc_global_polls/v1/poll`. The 'Tests' tab is selected, displaying the following test script:

```
1 tests["Status code is 201"] = responseCode.code === 201;
2 tests["Location Header is present"] = postman.getResponseHeader("Location");
3 tests["Content-Type is application/json;charset=UTF-8"] = postman.getResponseHeader
  ("Content-Type") === "application/json;charset=UTF-8";
4 tests["Response Body Contains number"] = responseBody.has("number");
5 tests["Response Body Contains name"] = responseBody.has("name");
```

The results section shows 5/5 tests passed:

- PASS Status code is 201
- PASS Location Header is present
- PASS Content-Type is application/json;charset=UTF-8
- PASS Response Body Contains number
- PASS Response Body Contains name

On the right side, a sidebar titled 'SNIPPETS' lists various assertions and tools:

- Clear a global variable
- Clear an environment variable
- Response body: Contains string
- Response body: Convert XML body to a JSON Object
- Response body: Is equal to a string
- Response body: JSON value check
- Response headers: Content-Type header check
- Response time is less than 200ms
- Set a global variable

Create New Resource in Polls API – Edit poll

The ‘Edit poll’ resource will be used to modify an existing poll record in the ‘Polls’ application.

16. Open the Polls API in studio and click New on the Resources related list to create a new resource.

17. Specify the following properties for the new resource.

Name: **Edit poll**

API Version: **v1**

HTTP method: **PATCH**

Relative path: **{poll_id}**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab3_edit_poll

The screenshot shows the 'Scripted REST Resource' configuration screen. The 'Name' field is set to 'Edit poll'. The 'API version' field is set to 'v1'. The 'HTTP method' is set to 'PATCH'. The 'Relative path' is set to '{poll_id}'. The 'Resource path' is shown as '/api/x_snc_polls/v1/poll/{poll_id}'. The 'Request routing' section provides a detailed description of how the route configuration specifies the HTTP method and relative path. The 'Active' checkbox is checked.

Click **Submit**.

Test with REST API Explorer

Open 'Edit poll' resource and Click **Explore REST API** in related actions.

18. **Edit poll** resource is preselected in API Explorer.

The screenshot shows the REST API Explorer interface. On the left, there are dropdown menus for Namespace (x_snc_polls), API Name (Poll), and API Version (v1). Below these are three buttons: 'Create new poll (POST)', 'Retrieve poll details (GET)', and 'Edit poll (PATCH)'. The 'Edit poll (PATCH)' button is highlighted with a red box. On the right, under the 'Poll' section, a 'PATCH' request is displayed with the URL `http://10.11.91.87:16001/api/x_snc_polls/v1/poll/{poll_id}`. The 'Prepare request' tab is active, showing a path parameter 'poll_id' with the value 'ddee64b9443a1200964fac543127a1ab' highlighted with a yellow box and a red arrow pointing to it from the left side of the screen.

a. Fill in request body in raw tab under Request body section.

A sample request payload can be found at:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_edit_poll_sample_request

b. Copy the sample payload into the 'Raw' tab.

The screenshot shows the 'Request Body' section of the API Explorer. It has tabs for 'Builder' (which is selected) and 'Raw'. The 'Builder' tab displays a JSON payload:

```
[{"question": "what is your favorite film", "choices": [{"choice": "Avatar"}, {"choice": "Avengers"}, {"choice": "Toy Story"}, {"choice": "Frozen"}]}]
```

 Below the builder is a 'Send' button. At the bottom, there are links to generate code snippets in various languages: [ServiceNow Script] [cURL] [Python] [Ruby] [JavaScript] [Perl] [Powershell].

Click **Send**.

19. Verify response status code is **204-No content**.

The screenshot shows the 'Response' section of Postman. It includes a table for Headers and a large text area for the Response Body.

Headers	
Content-Encoding	gzip
Date	Sun, 17 Apr 2016 16:17:06 GMT
Server	ServiceNow
X-Is-Logged-In	true

Response Body

```
""
```

Create tests in Postman

20. In Postman select the 'CC17:Edit poll' request. This is a pre-built request that already contains an appropriately formatted payload for calling the 'Edit poll' resource.

The screenshot shows the 'SCRIPTED REST API WORKSHOP' collection in Postman. A specific request, 'PATCH CC17: Edit poll', is highlighted with a grey background.

- GET CC17: Intro Lab
- GET CC17: Retrieve poll detail
- GET CC17: Retrieve poll detail w/ TEST
- PATCH CC17: Edit poll**
- PATCH CC17: Edit poll w/ TEST
- POST CC17: Create new poll
- POST CC17: Create new poll w/ TEST
- DEL CC17: Delete a poll
- DEL CC17: Delete a poll w/ TEST
- POST CC17: Vote in poll
- POST CC17: Vote in poll w/ TEST
- GET CC17: Retrieve poll results

21. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. Once you've updated those values save and then send the request.

As you saw when you tested with the REST API Explorer a successful response will include a **204** status code and an empty payload. Let's add tests in Postman that verify that the following details in the response:

- Response status code is 204
- Response payload is empty

22. Update the request in Postman to include the following:

```
tests["Status code is 204"] = responseCode.code === 204;  
tests["Body is empty"] = responseBody === "";
```

For ease you can also copy these from:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_edit_poll_test_script

23. After adding the tests above save your request in Postman and Send the request. In the response you should see the following tests and results.

The screenshot shows a Postman request configuration for a PATCH method. The URL is set to `https://api.servicenow.com/v1/poll/dad383c41`. The 'Tests' tab is active, displaying the following test script:

```
1 tests["Status code is 204"] = responseCode.code === 204;  
2 tests["Body is empty"] = responseBody === "";
```

The 'Results' tab shows the test outcomes:

- PASS Status code is 204
- PASS Body is empty

A sidebar on the right provides a list of available test snippets, including:

- Clear a global variable
- Clear an environment variable
- Response body: Contains string
- Response body: Convert XML body to a JSON Object
- Response body: Is equal to a string
- Response body: JSON value check
- Response headers: Content-Type header check
- Response time is less than 200ms
- Set a global variable

Create New Resource in Polls API – Vote in poll

24. The ‘Vote in poll’ resource will be used to cast a vote for an answer to a specific question or set of questions that are part of a poll in the ‘Polls’ application. Open Polls API in studio and add a **Resource** to the API. Click **New** on the **Resources** related list.

The screenshot shows the ServiceNow API Studio interface for creating a new API resource. The top section displays basic configuration details:

- API definition:** Poll
- Name:** Vote in poll (highlighted with a red box)
- Application:** Polls
- API version:** v1 (highlighted with a red box)
- Active:** checked

The middle section, titled "Request routing", contains the following fields:

- HTTP method:** POST (highlighted with a red box)
- Relative path:** /{poll_id}/vote (highlighted with a red box)
- Resource path:** /api/x_snc_polls/v1/poll/{poll_id}/vote

The bottom section, titled "Implement the resource", shows a script editor with the following code:

```
1 * (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     var pollId = request.pathParams.poll_id;  
3     var pollHelper = new x_snc_polls.PollData_Creator();  
4     // Validate if poll record exists  
5     var pollRecord = new GlideRecord("x_snc_polls_poll");  
6     pollRecord.query("poll_id = " + pollId);  
7     if (pollRecord.next()) {  
8         pollRecord.setBoolean("voted", true);  
9         pollRecord.update();  
10    }  
11    response.setBody(pollRecord);  
12}
```

Specify the following properties for the new resource.

Name: **Vote in poll**

API Version: **v1**

HTTP method: **POST**

Relative path: **{poll_id}/vote**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab3_vote_in_poll

The screenshot shows the 'API definition' screen for a 'Poll' API. The 'Name' field is set to 'Vote in poll' and the 'API version' is 'v1'. The 'Active' checkbox is checked. In the 'Request routing' section, the 'HTTP method' is 'POST' and the 'Relative path' is '{poll_id}/vote'. The 'Resource path' is '/api/x_snc_polls/v1/poll/{poll_id}/vote'. In the 'Implement the resource' section, there is a script editor containing the following code:

```

* Script
(function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
    var pollId = request.pathParams.poll_id;
    var pollHelper = new x_snc_polls.PollData_Creator();
    // Validate if poll record exists
    var pollRecord = new GlideRecord("x_snc_polls_poll");


```

NOTE: Observe the custom response string being written to the response using the 'getStreamWriter' method. The getStreamWriter method is used to produce a custom response in Scripted REST APIs and allows you (the API creator) to precisely specify the format of the response. It is important to set content type and status code if writing to stream directly.

The screenshot shows the script editor with the following code:

```

1 v
2
3     var pollId = request.pathParams.poll_id;
4     var pollHelper = new x_snc_polls.PollData_Creator();
5
6     // Validate if poll record exists
7     var pollRecord = new GlideRecord("x_snc_polls_poll");
8     pollRecord.get(pollId);
9
10    var voteData = request.body.data.votes;
11
12    // Record votes
13    pollHelper.voteInPoll(voteData, pollId);
14
15    // Set response details
16    response.setStatus(201);
17    response.setContentType("application/json");
18    var responseBody = '{"message": "Voting successful"}';
19    response.getWriter().writeString(responseBody);
20
21 })(request, response);


```

The lines from 16 to 19 are highlighted with a red box.

Click Submit.

Test with REST API Explorer

25. Open 'Vote in poll' resource and Click **Explore REST API** in related actions.
26. **Vote in poll** resource is preselected in API Explorer. Fill in request body in raw tab under Request body section.

The screenshot shows the REST API Explorer interface. On the left, there are dropdown menus for Namespace (x_snc_polls), API Name (Poll), and API Version (v1). Below these are several action buttons: 'Create new poll (POST)', 'Retrieve poll details (GET)', 'Edit poll (PATCH)', and 'Vote in poll (POST)'. The 'Vote in poll (POST)' button is highlighted with a red box. On the right, under the 'Poll' category, the 'Vote in poll - Answer poll' section is shown. It includes a POST method URL: `POST http://10.11.91.87:16001/api/x_snc_polls/v1/poll/{poll_id}/vote`. The 'Path parameters' section shows a table with one row: `poll_id` (Value: `ddee64b9443a1200964fac543127a1ab`). A red arrow points from the 'Value' field back up towards the 'Vote in poll (POST)' button.

A sample request payload can be found at:

http://bit.ly/CC17_ScriptedRESTAPI_Lab3_vote_in_poll_sample_request

NOTE: you will need to update the 'poll_id' to be that of a specific poll that exists in the Polls application on your lab instance.

The screenshot shows the 'Request Body' builder. It has tabs for 'Builder' (which is selected) and 'Raw'. The 'Builder' tab displays a JSON object: `{ "votes": [{ "question_id": "d5f383c4137612006ae13d62f244b056", "vote": "Yellow" }] }`. A red box highlights this JSON object. At the bottom, there is a 'Send' button and a link to 'ServiceNow Script'.

Click **Send**.

27. Verify response status code is **201 Created**.

Response

Status code	201 Created
Headers	
Content-Encoding	gzip
Content-Type	application/json
Date	Sun, 17 Apr 2016 17:27:27 GMT
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true

Response Body

```
{  
  "message": "Voting successful"  
}
```

Create tests in Postman

27. In Postman select the ‘CC17:Vote in poll’ request. This is a pre-built request that already contains an appropriately formatted payload for calling the ‘Vote in poll’ resource.
28. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. Once you’ve updated those values save and then send the request.
29. As you saw when you tested with the REST API Explorer a successful response will include a **201** status code and an JSON payload informing you that voting was successful. Add tests in Postman that verify that the following details in the response:
 - Response status code is 201
 - Response headers include Content-Type of application/json; charset=UTF-8
 - Response body contains the text: “Voting successful”

30. You are on your own to create these tests in Postman. You can refer back to the tests you've created in the previous steps for help.
31. Once you've added the tests save the request and send it. If you were successful you should see all the tests passing.

A screenshot of the Postman interface showing the 'Tests' tab selected. There are three test cases listed, each with a green 'PASS' status indicator:

- PASS Status code is 201
- PASS Content-Type is application/json; charset=UTF-8
- PASS Response Body Contains 'Voting successful'

Note: If you are really stuck here you can refer to the pre-built request in the Postman collection named "CC17: Vote in poll w/ TEST" to see this request with tests fully specified.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

32. Similar to creating the Lab3 starting branch, the completed lab can also be checked out from a tag (**Lab3-complete**) in Source control.

33. In **Studio**, navigate to **Source Control > Create Branch**.

34. In the pop-up window, enter a branch name, then select **Lab3-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab3-branch-complete**

Create from Tag: **Lab3-complete**

35. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

36. Verify Studio is on branch **my-Lab3-branch-complete**.

Lab 3 is complete. You are now ready to begin lab 4.

Lab 4 Enforcing Security

Lab Goal

In Lab 4 you'll continue building out the REST API for the **Polls** application adding resources to support retrieving the results of a poll which includes details of individuals votes as well as the ability to delete a poll. These operations expose functionality that should be restricted to users with an additional role so that we can limit access to see how individual users voted as well as be able to delete polls.

Scripted REST APIs allow you to specify ACLs that requestors must have to be able to make a request both at the API and Resource level. These ACLs can then be associated users or groups via the standard access control mechanism in ServiceNow.

Scripted REST APIs allow you to configure, at both the API and Resource level, if a requestor needs to **authenticate** (via Basic Auth or OAuth2.0) to ServiceNow to make requests. In addition, you can configure if the requestor must be authorized, via specific ACLs, to make a request to your API.

In building out these additional resources you will familiarize yourself with how you can use the security features of Scripted REST APIs to secure your REST API.

Create Lab 4 starting branch

1. In **Studio**, navigate to **Source Control > Create Branch**.
2. In the pop-up window, enter a branch name, then select **Lab4-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab4-branch**
Create from Tag: **Lab4-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab4-branch**.
5. You are now ready to start **Lab 4**.

Create New Resource in Polls API – Retrieve poll results

6. Open Polls API from studio. Add a **Resource** to the API. Click **New** on the **Resources** related list.

Give the resource a **Name**. Complete the script.

Name: **Retrieve poll results**

API Version: **v1**

HTTP method: **GET**

Relative path: **/{poll_id}/results**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab4_retrieve_poll_results

NOTE: Notice that the script is using the GlideRecordSecure API.

The screenshot shows the 'Scripted REST Resource' configuration page. The 'Name' field is set to 'Retrieve poll results'. The 'Application' dropdown is set to 'Polls' and 'API version' is set to 'v1'. The 'Request routing' section shows 'HTTP method' as 'GET' and 'Relative path' as '/{poll_id}/results'. The 'Resource path' is listed as '/api/x_snc_polls/v1/poll/{poll_id}/results'. In the 'Implement the resource' section, there is a script editor containing the following code:

```
1 v  (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2     var id = request.pathParams.poll_id;  
3     var pollHelper = new x_snc_polls.PollData_Retriever();  
4     var pollRecord = new GlideRecordSecure("x_snc_polls_poll");  
5     pollRecord.get(id);  
6 }
```

7. Enable ACL authorization on the resource by setting an ACL. ACL settings are available under Security tab

Requires ACL authorization: **checked**

ACLs: Click to unlock, and browse to select the **Poll Manager** ACL

Resources can specify security settings that override the parent settings.

By default resources 'Require authentication' but do not 'Require ACL authorization'. To make a resource public, meaning no authentication is required to access the resource, uncheck 'Requires authentication'.

To require authorization, select the 'Requires ACL authorization' check box and select an ACL record(s). Leave the 'ACL' field blank to enforce the 'Default ACLs' from the parent API. Access is granted if at least one matching ACL record is found.

[More info](#)

Requires authentication

Requires ACL authorization

ACLs

NOTE: Only ACLs of type REST Endpoint can be used.

Click **Submit**.

BEST PRACTICES

Do: Use the *GlideRecordSecure API* in your Scripted REST API Resource scripts to ensure that you are enforcing existing access controls on the requesting user when interacting with ServiceNow records.

Do: Test your access controls, both Authentication and Authorization, before making your API available to consumers.

Test with REST API Explorer

8. Open 'Retrieve poll results' resource and Click **Explore REST API** in related actions.
9. **Retrieve poll results** resource is preselected in API Explorer. Fill in sys_id of poll.

REST API Explorer

Namespace: x_snc_polls

API Name: Poll

API Version: v1

Poll

Retrieve poll results

GET http://10.11.91.87:16001/api/x_snc_polls/v1/poll/{poll_id}/results

Create new poll (POST)
Retrieve poll details (GET)
Delete poll (DELETE)
Edit poll (PATCH)
▶ Retrieve poll results (GET)
Vote in poll (POST)

Prepare request

Path parameters

Name	Value
* poll_id	ddee64b9443a1200964fac543127a1ab

Query parameters

Add query parameter

Click **Send**.

10. Verify response status code is **200-Ok**.

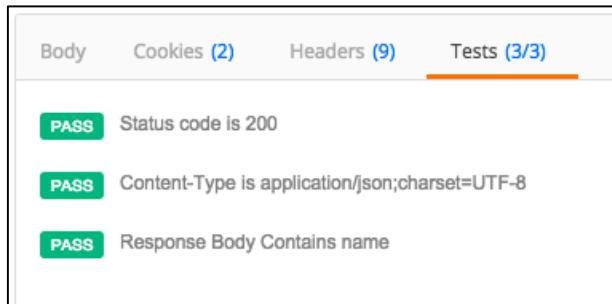
Response

Status code	200 OK
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 17:51:53 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "result": { "name": "First Poll", "questions": [{ "question": "Favorite number", "users": [] }, { "question": "Favorite color", "users": [{ "user": "admin", "answer": "Yellow" }] }] } }	

Create tests in Postman

11. In Postman select the ‘CC17: Retrieve poll results’ request. This is a pre-built request that will make a request to the ‘Retrieve poll results’ resource.
12. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. Once you’ve updated those values save and then send the request. If the request is successful (200 OK) you will see a response similar to the one you saw when testing in the REST API Explorer.
13. Now that you’ve made a successful request add tests to your Postman requests to validate the request matches the expected results. Add tests that verify the following details in the response:
 - Response status code is 200
 - Response headers include Content-Type of application/json;charset=UTF-8
 - Response body contains the text: “name”
14. You are on your own to create these tests in Postman. You can refer back to the tests you’ve created in the previous exercises for help.

15. Once you've added the tests save the request and send it. If you were successful you should see all the tests passing.



A screenshot of the Postman interface showing test results. The tabs at the top are Body, Cookies (2), Headers (9), and Tests (3/3). The Tests tab is selected and highlighted with an orange underline. Below the tabs, there are three test cases listed, each with a green 'PASS' status indicator:

- PASS Status code is 200
- PASS Content-Type is application/json;charset=UTF-8
- PASS Response Body Contains name

Note: If you are really stuck here you can refer to the pre-built request in the Postman collection named “CC17: Retrieve poll results w/ TEST” to see this request with tests fully specified.

Create New Resource in Polls API – Delete poll

16. Open Polls API from studio. Add a **Resource** to the API. Click **New** on the **Resources** related list.

17. Give the resource a **Name**. Complete the script.

Name: **Delete poll**

API Version: **v1**

HTTP method: **DELETE**

Relative path: **/{poll_id}**

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab4_delete_poll

The screenshot shows the 'Scripted REST Resource' configuration page. The 'Name' field is set to 'Delete poll'. The 'Application' dropdown is set to 'Polls' and the 'API version' is 'v1'. The 'HTTP method' is 'DELETE' and the 'Relative path' is '/{poll_id}'. The 'Resource path' is '/api/x_snc_polls/v1/poll/{poll_id}'. The 'Script' tab displays the following JavaScript code:

```
1 v  (function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {  
2 v   var pollId = request.pathParams.poll_id;  
3 v   var pollRecord = new GlideRecordSecure("x_snc_polls_poll");  
4 v   pollRecord.get(pollId);  
5 v   if (pollRecord.isValidRecord()) {  
6 v     pollRecord.deleteRecord();  
7 v   }  
8 v } );
```

18. Enable ACL authorization on the resource by setting an ACL. ACL settings available under Security tab.

Requires ACL authorization: **checked**

ACLs: Click to unlock, and browse to select the **Poll Manager** ACL

The screenshot shows the 'Access Control' page with the 'Security' tab selected. A red box highlights the 'Requires ACL authorization' checkbox, which is checked. Another red box highlights the 'ACLs' button, which is currently set to 'Poll Manager'. Below these fields is a note: 'To require authorization, select the 'Requires ACL authorization' check box and select an ACL record(s). Leave the 'ACL' field blank to enforce the 'Default ACLs' from the parent API. Access is granted if at least one matching ACL record is found.'

NOTE: Only ACLs of type REST Endpoint can be used

Click **Submit**.

The screenshot shows the 'Access Control' page for the 'Poll Manager' ACL. A red box highlights the 'Type' field set to 'REST_Endpoint'. Another red box highlights the 'Name' field set to 'Poll Manager'. The 'Requires role' section shows a single role assigned: 'x_snc_polls.poll_manager'. A red box highlights the 'Scripted REST Resources' table, which lists two resources: 'Delete poll' (DELETE method) and 'Retrieve poll results' (GET method), both associated with the 'Poll' API definition and 'v1' version.

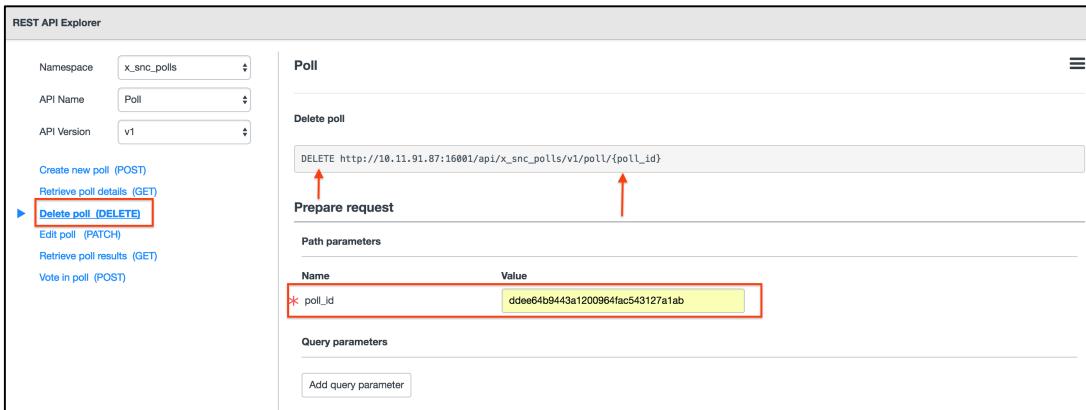
Name	HTTP method	Active	Relative path	API definition	API version
Delete poll	DELETE	true	/[poll_id]	Poll	v1
Retrieve poll results	GET	true	/[poll_id]/results	Poll	v1

NOTE: REST_Endpoint type ACLs (as shown above) are used to restrict access to Scripted REST API Resources. The 'Poll Manager' ACL has been specified on the 'Delete poll' resource and restricts access to this resource to users who have the role 'x_snc_polls.poll_manager'.

Only users with this role can make requests to the 'Delete poll' resource.

Test with REST API Explorer

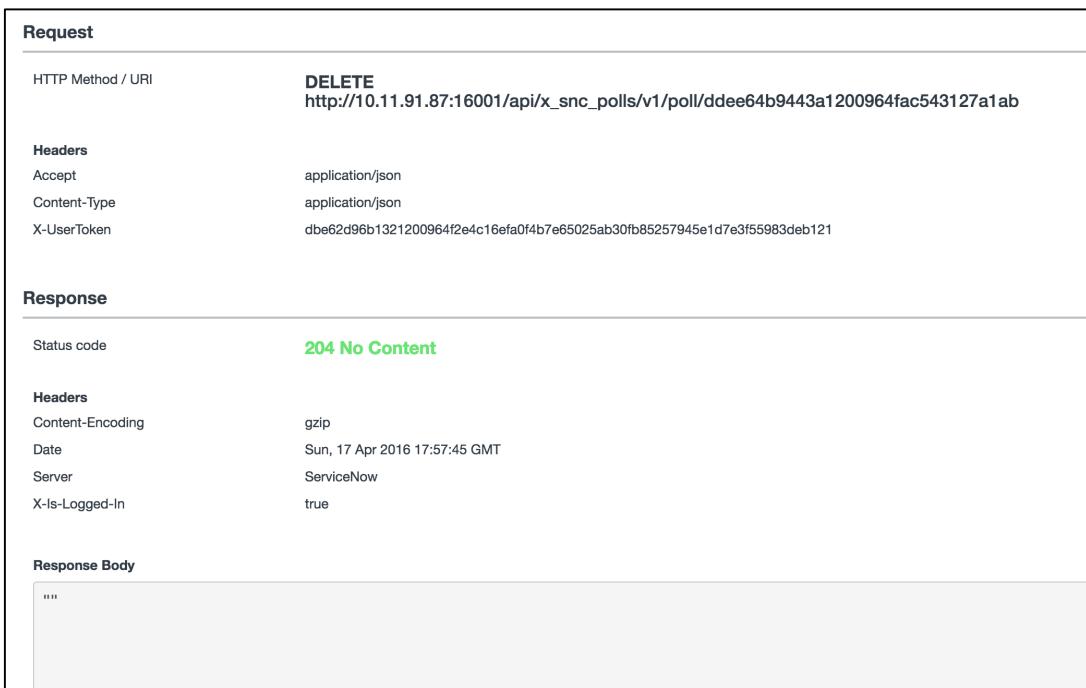
19. Open Delete poll resource and click **Explore REST API** in related actions.
20. **Delete poll** resource is preselected in API Explorer. Fill in request body in raw tab under Request body section.



The screenshot shows the REST API Explorer interface. On the left, there's a sidebar with various API actions: Create new poll (POST), Retrieve poll details (GET), Delete poll (DELETE) (which is highlighted with a red box), Edit poll (PATCH), Retrieve poll results (GET), and Vote in poll (POST). The main panel is titled 'Poll' and shows the 'Delete poll' action. The request URL is displayed as 'DELETE http://10.11.91.87:16001/api/x_snc_polls/v1/poll/{poll_id}'. Below the URL, there's a 'Prepare request' section with 'Path parameters' and 'Query parameters'. A 'Name' column has 'poll_id' listed, and a 'Value' column has 'ddee64b9443a1200964fac543127a1ab'. The 'poll_id' entry is highlighted with a red box and a yellow background.

Click **Send**.

21. Verify response status code is **204-No content**.



The screenshot shows the REST API Explorer interface with the 'Request' tab selected. It displays the HTTP method (DELETE) and URL (http://10.11.91.87:16001/api/x_snc_polls/v1/poll/ddee64b9443a1200964fac543127a1ab). The 'Headers' section shows Accept: application/json, Content-Type: application/json, and X-UserToken: dbe62d96b1321200964f2e4c16efa0f4b7e65025ab30fb85257945e1d7e3f55983deb121. The 'Response' tab is selected and shows the status code as '204 No Content'. The 'Headers' section of the response includes Content-Encoding: gzip, Date: Sun, 17 Apr 2016 17:57:45 GMT, Server: ServiceNow, and X-Is-Logged-In: true. The 'Response Body' section is empty, containing only three dots (...).

Create tests in Postman

22. In Postman select the 'CC17:Delete poll' request. This is a pre-built request that will make a request to the 'Delete poll' resource.

23. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Use your admin credentials for this request. And make sure that the admin user has the ‘x_snc_polls.poll_manager’ role. Once you’ve updated those values save and then send the request. If the request is successful (204 No Content) you will see a response similar to the one you saw when testing in the REST API Explorer.

24. Add tests that verify the following details in the response:

- Response status code is 204
- Response body is empty

25. After adding these tests issue the request and verify that your tests are passing as shown below.

Body	Cookies (2)	Headers (5)	Tests (2/2)
			PASS Status code is 204
			PASS Response Body is empty

26. Create a new poll in your instance and then update this request in Postman to use the new poll id and update the user credentials to use a user that **does not have** the ‘x_snc_polls.poll_manager’ role. Update the request in Postman and send the request.

27. **NOTE:** you have been deleting polls so you may need to go back and create some additional poll records in your instance so that there are polls that you can delete (hint use insert and stay to quickly create new polls for testing).

28. Send your updated request now and verify that for a user that when making a request with a user that **does not have** the ‘x_snc_polls.poll_manager’ role you receive a status code of **403 Forbidden** and that your test fail case ‘Status code is 204’ **fails** as shown below.

Body	Cookies (2)	Headers (10)	Tests (1/2)
			FAIL Status code is 204
			PASS Response Body is empty

29. **NOTE:** If you are really stuck here you can refer to the pre-built request in the Postman collection named “CC17: Delete a poll w/ TEST” to see this request with tests fully specified.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

30. Similar to creating the Lab4 starting branch, the completed lab can also be checked out from a tag (**Lab4-complete**) in Source control.
31. In **Studio**, navigate to **Source Control > Create Branch**.
32. In the pop-up window, enter a branch name, then select **Lab4-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab4-branch-complete**

Create from Tag: **Lab4-complete**

33. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

34. Verify Studio is on branch **my-Lab4-branch-complete**.

35. Close the REST API Explorer and API Analytics dashboard windows.

Lab 4 is complete. You are now ready to begin lab 5.

Lab Goal

Versioning a REST API is a common task when you want to introduce new functionality or behaviors to your REST API but don't want to break existing clients. Scripted REST APIs support easily versioning your resources. With versioning support you can quickly create new versions of existing resources to introduce new functionality. You have the ability to specify what version of a resource is the default and to which requests will be routed if the client does not specify a version in the URL or to force the clients of your REST API to include a version in the URL they make requests to.

In this lab you will add a new version to the Polls REST API you have been creating to familiarize yourself with the versioning functionality in Scripted REST APIs.

Lab 5 Versioning

BEST PRACTICES

Do: Version your REST API. By default, you do not need to create a version when creating a Scripted REST API in ServiceNow. Best practice is to version your REST API and disable the default route so that consumers must explicitly include the version number in their request URL. In this way you allow clients to decide if and when they want to use a later version of your REST API. If you need to force them to move to a new version at a later point in time you have the ability to disable versions.

Create Lab 5 starting branch

1. In Studio, navigate to Source Control > Create Branch.
2. In the pop-up window, enter a branch name, then select **Lab5-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab5-branch**

Create from Tag: **Lab5-start**

3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab5-branch**.
5. You are now ready to start **Lab 5**.

Add Version to Poll REST API

6. Open Poll API and click Add New version.

The screenshot shows the 'Scripted REST Service' configuration for the 'Poll' API. The 'Name' is set to 'Poll' and 'API ID' to 'poll'. The 'Active' checkbox is checked. The 'Application' is 'Polls' and the 'API namespace' is 'x_snc_polls'. The 'Base API path' is '/api/x_snc_polls/poll'. Below these fields, there are tabs for 'Security', 'Versioning', 'Content Negotiation', and 'Documentation'. A note about Default ACLs states: 'Default ACLs may be selected to apply to all resources, but individual resources can override this setting.' It also specifies that 'The Default ACLs are enforced for a resource when: • The resource 'Requires authentication' and 'Requires ACL authorization' fields are selected, and • The resource itself does not reference any ACL records'. A link to 'More info' is provided. Under 'Default ACLs', there is a lock icon. At the bottom, there are 'Update' and 'Delete' buttons. A 'Related Links' section includes 'Add new version', 'Explore REST API', and 'API analytics', with 'Add new version' being highlighted by a red box.

7. Select version 1 to copy resources. Click Ok.

The screenshot shows the 'Add new version' dialog. It has a 'Cancel' button on the left and an 'OK' button on the right. A checkbox labeled 'Make this version the default' is highlighted with a red box. To its right, a dropdown menu showing 'v1' is also highlighted with a red box. The 'OK' button is a large blue button at the bottom right.

To add a new version, use the 'Add new version' link below. You can select one version as the default. Clients can access the default version using either the versioned or non-versioned URI path.

Versions may also be inactivated or deprecated:

- Resources belonging to inactive versions cannot serve requests
- Resources belonging to deprecated versions can serve requests, but are identified as 'Deprecated' in documentation

[More info](#)

Default version: No active default version

Version ID	Is default	Active	Deprecated
v2	false	true	false
v1	false	true	false

[Insert a new row...](#)

NOTE: Both versions of the API have default set to **false**. This means that clients consuming this API must include the resource version in the URL.

Resources (12) Request Headers Query Parameters

Resources New Go to Name Search

API definition = Poll

Name	HTTP method	Relative path	Resource path	API version	Active
Create new poll	POST	/	/api/x_snc_polls/v1/poll	v1	true
Create new poll (v2)	POST	/	/api/x_snc_polls/v2/poll	v2	true
Delete poll	DELETE	/{poll_id}	/api/x_snc_polls/v1/poll/{poll_id}	v1	true
Delete poll (v2)	DELETE	/{poll_id}	/api/x_snc_polls/v2/poll/{poll_id}	v2	true
Edit poll	PATCH	/{poll_id}	/api/x_snc_polls/v1/poll/{poll_id}	v1	true
Edit poll (v2)	PATCH	/{poll_id}	/api/x_snc_polls/v2/poll/{poll_id}	v2	true
Retrieve poll details	GET	/{poll_id}	/api/x_snc_polls/v1/poll/{poll_id}	v1	true

NOTE: Observe every resource in v1 is copied and added to v2

Test with Postman

8. In Postman, review the requests named ‘CC17: Create new poll v1’ and ‘Create new poll v2’ noting the version number is explicitly specified in the URL for these two requests.
9. Add the tests that you added to the ‘CC17: Create new poll’ request to this the V1 and V2 requests. The behavior between the V1 and V2 resources has not been updated so you can copy and paste your test cases from the ‘CC17: Create new poll’ request and the tests should pass.
10. Verify that your test cases pass successfully.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

11. Similar to creating the Lab2 starting branch, the completed lab can also be checked out from a tag (**Lab5-complete**) in Source control.
12. In **Studio**, navigate to **Source Control > Create Branch**.
13. In the pop-up window, enter a branch name, then select **Lab5-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab5-branch-complete**

Create from Tag: **Lab5-complete**

14. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
15. Verify Studio is on branch **my-Lab5-branch-complete**.
16. Close the REST API Explorer and API Analytics dashboard windows.

Lab 5 is complete. You are now ready to begin lab 6.

Lab Goal

Errors... they happen to the best of us. Whether you are making requests to a 3rd party REST API or your own REST API there are times when you receive errors. Ideally the error message provides you (the client) with enough information to realize what went wrong, if it was your fault (client) or their fault (REST API) and how you can proceed.

Scripted REST APIs provide a helper API (`sn_ws_err`) to make it easier for you as the REST API designer to easily return consistent and informative error messages from your REST API.

Lab 6 Error handling

Create Lab 6 starting branch

1. In **Studio**, navigate to **Source Control > Create Branch**.
2. In the pop-up window, enter a branch name, then select **Lab6-start** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab6-branch**
Create from Tag: **Lab6-start**
3. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.
4. Verify Studio is on branch **my-Lab6-branch**.
5. You are now ready to start **Lab 6**.

Add Error handling to API – Retrieve poll detail

6. Open Retrieve poll detail (v2).
7. Modify script to check if poll record exists and send a 404 error response.

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab6_retrieve_poll_detail_v2

```
1 * (function process( /*RESTAPIrequest*/ request, /*RESTAPIResponse*/ response) {
2     var id = request.pathParams.poll_id;
3     var pollHelper = new x_snc_polls.PollData_Retriever();
4     var pollRecord = new GlideRecordSecure("x_snc_polls_poll");
5     pollRecord.get(id);
6     if (!pollRecord.isValidRecord()) {
7         throw new sn_ws_err.NotFoundError("Poll not found");
8     }
9     var pollResponse = {
10        name: pollRecord.getValue("name"),
11        questions: pollHelper.getQuestions(id).as_list(),
12    };
13    return pollResponse;
14 });
15 })(request, response);
16 })
```

Test with REST API Explorer

8. Open ‘Retrieve poll detail (v2)’ resource and Click **Explore REST API** in related actions.
9. Version v2 of Retrieve poll results resource is preselected in API Explorer.

Namespace: x_snc_polls

API Name: Poll

API Version: v2

Create new poll (v2) (POST)

▶ **Retrieve poll details (v2) (GET)**

Delete poll (v2) (DELETE)

Edit poll (v2) (PATCH)

Retrieve poll results (v2) (GET)

Vote in poll (v2) (POST)

Poll

Retrieve poll details (v2) - Retrieve poll, questions, choices and votes polled by sysid

GET http://10.11.91.87:16001/api/x_snc_polls/v2/poll/{poll_id}

Prepare request

Path parameters

Name	Value
* poll_id	

Query parameters

10. Fill in **invalid** sys_id of poll and click Send.
11. Verify response status code is **404-Not Found**.

Response	
Status code	404 Not Found
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 18:35:47 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
<pre>{ "error": { "detail": "", "message": "Poll not found" }, "status": "failure" }</pre>	

Create tests in Postman

12. In Postman select the ‘CC17: Retrieve poll detail V2’ request’. This is a pre-built request that will make a request to the ‘Retrieve poll detail V2’ resource.
13. Update the request replacing the {{instance_url}}, authorization credentials, and {{poll_id}} with values appropriate for your lab instance. Be sure to specify an **invalid** poll_id. Send a request and verify that you receive a **404 Not Found** status code and that the response body contains the same error message you received in REST API Explorer.
14. Add tests that verify the following details in the response:
 - Response status code is 404
 - Response body contains “error”
 - Response headers include Content-Type of application/json;charset=UTF-8
 - Response body is JSON and contains a status property with a value of ‘failure’.

NOTE: Postman has a test feature that allows you to parse a JSON response and verify it contains a specific property and value. See if you can figure out how to use it. More info can be found at https://www.getpostman.com/docs/testing_examples.

If you are stuck here you can refer to the pre-built request in the Postman collection named “CC17: Retrieve poll detail V2 w/ TEST” to see this request with tests fully specified.

Add Error handling to API – Vote in poll

15. Open Vote in poll(v2).

16. Modify script to check whether poll record exists as well as to see user already voted for the poll.

Script: Copy script from http://bit.ly/CC17_ScriptedRESTAPI_Lab6_vote_in_poll_v2

```
function process( /*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {
    var pollId = request.pathParams.poll_id;
    var pollHelper = new x_snc_polls.PollData_Creator();

    // Validate if poll record exists
    var pollRecord = new GlideRecordSecure("x_snc_polls_poll");
    pollRecord.get(pollId);
    if(!pollRecord.isValidRecord()){
        throw new sh_ws_err.NotFoundError("Poll Not found");
    }

    var voteData = request.body.data.votes;

    // Verify if already voted
    var pollResponseRecord = new GlideRecordSecure("x_snc_polls_poll_response");
    pollResponseRecord.addQuery("poll", pollId);
    pollResponseRecord.addQuery("sys_created_by", gs.getUserName());
    pollResponseRecord.query();
    if(pollResponseRecord.next()){
        throw new sh_ws_err.ConflictError("Already voted");
    }

    // Record votes
    pollHelper.voteInPoll(voteData, pollId);

    // Set response details
    response.setStatus(201);
}(request, response);
```

Test with REST API Explorer

Open 'Vote in poll (v2)' resource and Click **Explore REST API** in related actions.

17. Version v2 of Vote in poll resource is preselected in API Explorer.

18. Fill in an **invalid sys_id** of poll and click Send

19. Verify response status code is **404-Not Found**

Response

Status code	404 Not Found
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 18:35:47 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "error": { "detail": "", "message": "Poll not found" }, "status": "failure" }</br >	

20. Now fill in a **valid sys_id** and request body and click Send to vote.

Specify the request body as shown below. Sample script available for you to copy at:
http://bit.ly/CC17_ScriptedRESTAPI_Lab6_vote_in_poll_v2_sample_request

Request Body

Builder Raw

```
{  
  "votes": [  
    {"question_id": "ab4fe4b9443a1200964fac543127a1eb",  
     "vote": "Yellow"}  
  ]  
}
```

21. Verify response status code is **409-Conflict**.

NOTE: Depending on state of instance, you might need to fire the request twice. The first request is a valid vote while the second request results in a conflict response.

Response	
Status code	409 Conflict
Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Sun, 17 Apr 2016 19:12:31 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Transfer-Encoding	chunked
X-Is-Logged-In	true
Response Body	
{ "error": { "detail": "", "message": "Already voted" }, "status": "failure" }</br >	

Create tests in Postman

22. In Postman select the ‘CC17: Vote in poll V2’ request’. This is a pre-built request that will make a request to the ‘Vote in poll V2’ resource.

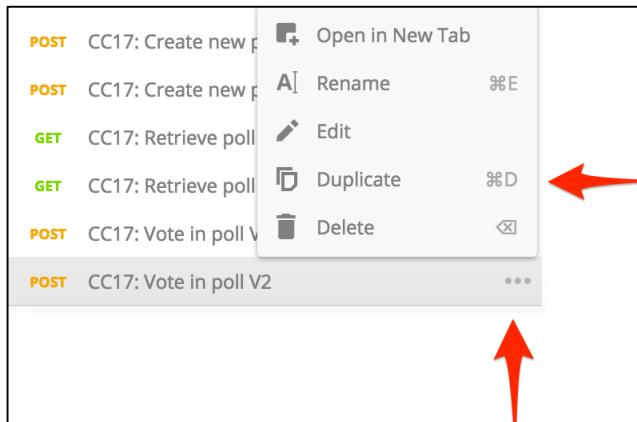
23. Update the request replacing the {{instance_url}}, authorization credentials, {{poll_id}}, and body of the request with vote details (question_id and vote value) appropriate for your lab instance.

NOTE: Depending on the state of instance, you might need to fire the request twice. The first request is a valid vote while the second request results in a conflict response. This is because you have added a constraint that users can only vote for a question once.

24. Add tests that verify the following details in the response:

- Response status code is 409 Conflict
- Response body contains ‘error’
- Response headers include Content-Type of ‘application/json; charset=UTF-8’
- Response body is JSON and contains an error property with message property with a value of ‘Already voted’.

25. In Postman create a copy of this request by clicking on the ‘...’ icon to the right of the request name in the left-hand list of requests as shown below and then clicking ‘Duplicate’. This will create a copy of your request that you will update to verify the error message that is returned when you attempt to vote on the same question twice.



26. Update this new request adding tests that verify the following details in the response:

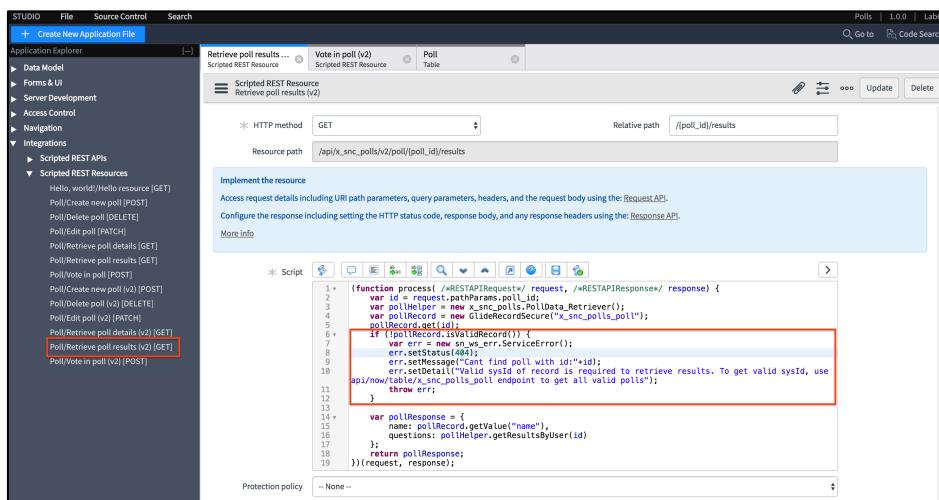
- Response status code is 409
- Response body contains “error”
- Response headers include Content-Type of application/json; charset=UTF-8
- Response body is JSON and contains a status property with a value of ‘failure’.

27. Issue the request and verify that all of your tests have completed successfully.

Add Error handling to API – Retrieve poll results

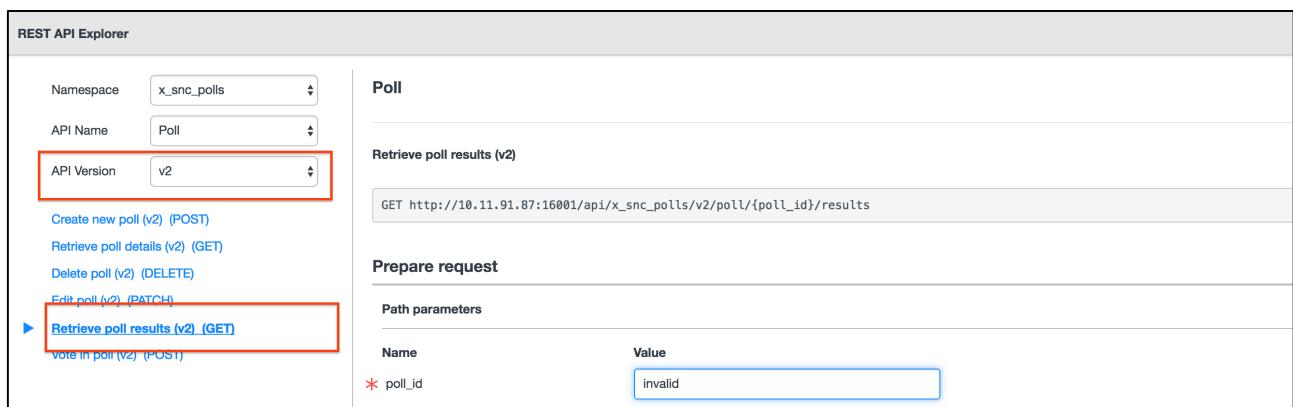
28. Open Retrieve poll results (v2).
29. Modify script to check if poll exists and send customized error response.

Script: Copy script: http://bit.ly/CC17_ScriptedRESTAPI_Lab6_retrieve_poll_results_v2



Test with REST API Explorer

30. Open 'Retrieve poll results (v2)' resource and Click **Explore REST API** in related actions.
31. Version v2 of **Retrieve poll results** resource is preselected in API Explorer.



32. Fill in invalid sys_id of poll and click Send.

33. Verify response status code is **404-Not Found** with custom Error message.

The screenshot shows the 'Response' section of Postman. The 'Status code' is highlighted in red as '404 Not Found'. The 'Headers' section lists various HTTP headers with their values. The 'Response Body' section contains a JSON object:

```
{  
  "error": {  
    "detail": "Valid sysId of record is required to retrieve results. To get valid sysId, use api/now/table/x_snc_polls_poll endpoint to get the sysId of the poll.",  
    "message": "Can't find poll with id:invalid"  
  },  
  "status": "failure"  
}
```

Create tests in Postman

34. Now that you have experience using Postman to build requests and create tests you are on your own for this last Postman test.
35. Create a new Postman request, either brand new or by duplicating an existing request, that makes a request to the **Retrieve poll results V2 resource** with an **invalid** poll id.
36. Add tests to this request verifying the following:
 - Response status code is 404
 - Response body contains “error”
 - Response headers include Content-Type of ‘application/json;charset=UTF-8’
 - Response body is JSON and contains an error property with a message child property that has a value of ‘Can’t find poll with id:invalid’.
37. Issue request and verify that all tests pass successfully.

Get Caught Up

If you were unable to successfully complete the lab this far, you can “fast forward” using the following steps. Otherwise proceed to the next section **Test with Postman**.

38. Similar to creating the Lab6 starting branch, the completed lab can also be checked out from a tag (**Lab6-complete**) in Source control.

39. In **Studio**, navigate to **Source Control > Create Branch**.

40. In the pop-up window, enter a branch name, then select **Lab6-complete** from the **Create from Tag menu**, and click **Create Branch**.

Branch: **my-Lab6-branch-complete**

Create from Tag: **Lab6-complete**

41. When the switch is complete, click **Close Dialog** in the Create Branch pop-up.

42. Verify Studio is on branch **my-Lab6-branch-complete**.

43. Close the REST API Explorer and API Analytics dashboard windows.

Lab 6 is complete. You are now ready to begin lab 7.

Lab Goal

Congratulations you've made it to the challenge lab. If you've made it this far you have either whizzed right through labs 1-6 and are a Scripted REST API and Postman expert or you're working on this challenge lab after the CreatorCon Workshop.

Throughout this workshop you have built a Scripted REST API for the Polls application and created requests with tests in the Postman collection that allow you to quickly issue requests to your REST API that also have test cases that verify the REST API is returning valid responses and behaving as intended.

Creating tests that validate your REST APIs behavior is a **BEST PRACTICE** that allows you to easily verify your REST API is behaving as intended and quickly identify any breaking changes in your REST API in the future as you make changes to add functionality or patch bugs.

Postman makes it easy to run these tests manually but **Nobody** enjoys running tests manually all the time. Wouldn't it be nice if you could automate these tests? I certainly think so. Let's take this one step further and see if you can get these tests to run from the command line. Postman provides a tool called **Newman** that allows you to run the tests in your existing Postman collection from the command line and see the results either at the command line or write them to a file. For this challenge lab your task is to install Newman and run your Postman collection from the command line. Check out the links below to download and install Newman and get more information on how to run a collection from the command line.

If you can get Newman running from the command line you are only a few steps away (bash, cron, PowerShell script) from automating your REST API test cases with Postman and Newman. This is outside the scope of this lab but you can imagine how you could integrate this with a continuous integration testing tool to have these tests run on a regular basis or as part of a build process.

Newman: <http://blog.getpostman.com/2014/05/12/meet-newman-a-command-line-companion-for-postman/>

Running collections from the command line:
https://www.getpostman.com/docs/newman_intro

Integrating automated API tests with Jenkins:
https://www.getpostman.com/docs/integrating_with_jenkins

Lab 7 Challenge Lab