REST Integrations

RESTful web services allow ServiceNow to interact with other applications. In this course you will configure ServiceNow to be both a web service provider and a web service consumer.

1. Outbound REST Integrations

In this module you will create an outbound REST Message. You will test the outbound REST Message then invoke the outbound REST Message from a script. You will populate a table field with a value read from the response body returned by the web service provider.

1.1 Outbound Integrations in ServiceNow Objectives

In this module you will learn to:

- Create an outbound REST Message
    - Endpoint
    - Header
    - Parameters
    - REST Method
- Test an outbound REST Message
    - Variable Substitutions
    - HTTP Status
    - Response Body
- Debug outbound REST Messages
    - Examine status and response content
    - Set log levels
    - Use Outbound HTTP Requests log
- Use Preview Script Usage to create server-side JavaScript for invoking the outbound REST Message
- Write server-side JavaScript to parse data out of the response body

1.2 ServiceNow as a Web Service Consumer

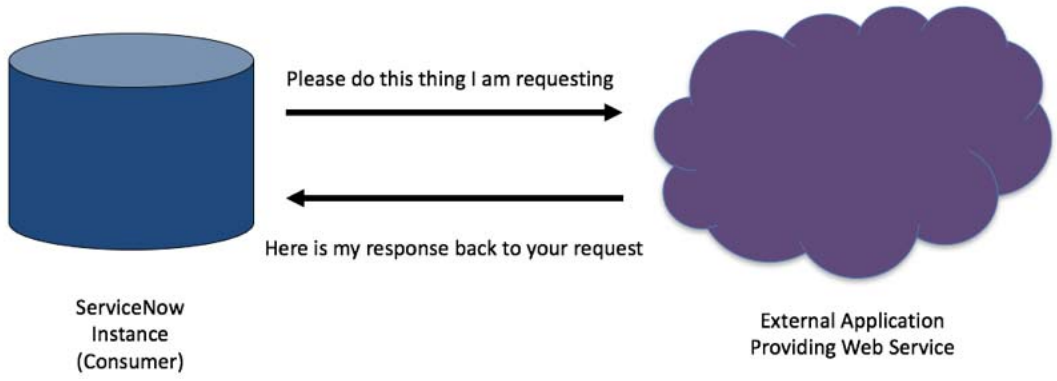
Web services make it possible for applications to connect to other software applications over a network allowing an exchange of information between the provider (server) and client (consumer).

A web service consumer (client) requests information from a web service provider (server). A web service provider processes the request and returns a status code and a response body. When the response body is returned, the web service consumer extracts information from the response body and takes action on the extracted data.

ServiceNow is able to consume web services from third party providers or from another ServiceNow instance.

Third party web services can provide information like:

- Stock information
- Geolocation coordinates
- Financial data
- Inventory
- Weather data
- Language translation



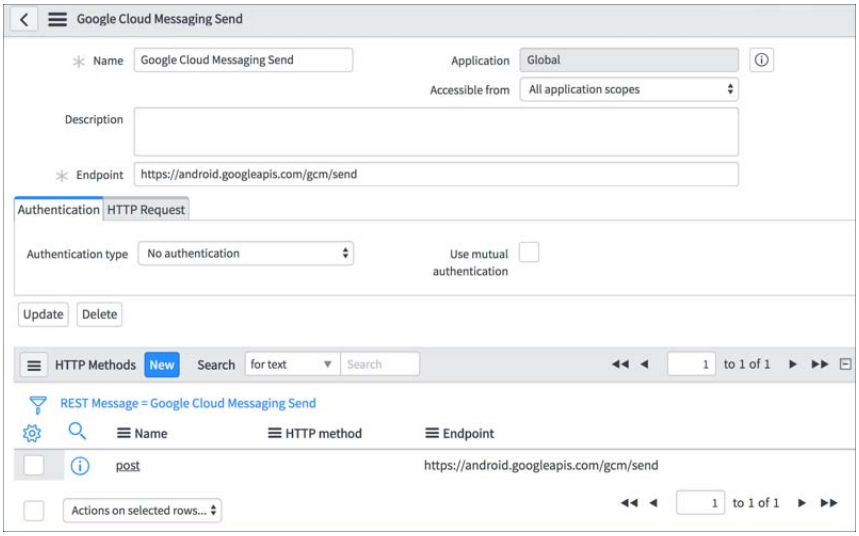
1.3 Outbound REST Messages

ServiceNow stores details on how to interact with external web services through REST in a REST Message record. The REST Message record includes::

- Endpoint
- Authentication
- HTTP Headers
- HTTP Method

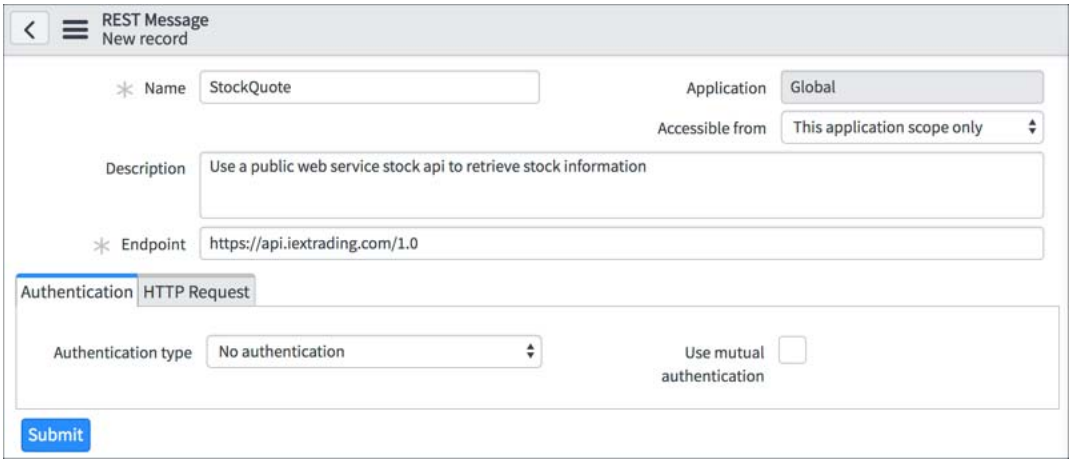
Outbound REST Messages allow developers to test web services and view the response body.

Once an outbound REST Message is configured and tested, the outbound REST Message can be invoked from any server-side script.



1.4 Creating an Outbound REST Message

To create an outbound REST Message, use the Application Navigator to open **System Web Services > Outbound > REST Message** or add a REST Message in Studio.



- **Name**: A descriptive name for the REST message. This value is used when invoking the Outbound REST Message from a script.
- **Endpoint**: Enter the endpoint that this REST message is sent to. The endpoint value may include variables using the format ${variable}.
- **Authentication type**: Select the type of authentication to use, if any, and the profile record that contains the user credentials. Outbound REST supports basic authentication and OAuth 2.0. Authentication configured here is inherited by the associated HTTP methods. You can configure authentication for each method which overrides any authentication setting at the message level.
- **Use mutual authentication**: Select to require both the web service provider and consumer to authenticate with each other before communicating. Outbound REST supports mutual authentication with basic authentication only.
- **HTTP Headers**: Double-click a row in the HTTP Headers embedded list to define the header Name and Value. The web service provider determines which headers are supported or required. See **List of HTTP Header Fields** (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) for a list of HTTP header fields.

1.5 Authenticating in an Outbound REST Message

Different web service providers may require a specific type of authentication. Outbound REST supports the following authentication formats.

- No authentication
- Basic authentication using a username and password
- OAuth 2.0 using an OAuth provider and profile

No Authentication

Many public web services are intended for the free, unauthenticated distribution of information from the web service. For public web services which do not require authentication, set the Authentication type field to **No authentication**.

Basic

The Basic authentication type passes a username and a password to the web service. Set the Authentication type field value to **Basic** then select a Basic auth profile.

A Basic auth profile consists of a name, a user, and a password.

To edit a Basic auth profile, type sys_auth_profile_basic.list into the Application Navigator then open a record for editing.

OAuth

OAuth is an Internet standard for giving users access to APIs without giving them a password. See the **How to setup OAuth2 authentication for RESTMessageV2 integrations** (https://community.servicenow.com/community?id=community_blog&sys_id=a1fce2a5dbd0dbc01dcaf3231f961939) blog for information about setting up OAuth.

Mutual Authentication

Mutual authentication requires the web service provider and consumer to authenticate with each other before communicating. Mutual authentication a is protocol/socket-level authentication compared to other authentication options which are application-level authentications. Mutual auth can be used in conjunction with other authentication mechanisms. To enable mutual authentication, select the **Use mutual authentication** option then select a Mutual authentication profile.

To define a **Mutual authentication profile** (https://docs.servicenow.com/bundle/kingston-application-development/page/integrate/outbound-web-services/task/t_CreateAProtocolProfile.html), use the Application Navigator to open **System Security > Protocol Profiles**.

This option is not available for web services which use a MID server.

1.6 Exercise: Create an Outbound REST Message

In this exercise, you will create an application called IPLookup. You will add an outbound REST Message to the application which uses the ipinfo.io public web service.

Preparation - View API Documentation

Visit the **ipinfo.io** (https://ipinfo.io/developers#specific-fields) and browse the documentation to see what is passed to the API and what is returned.

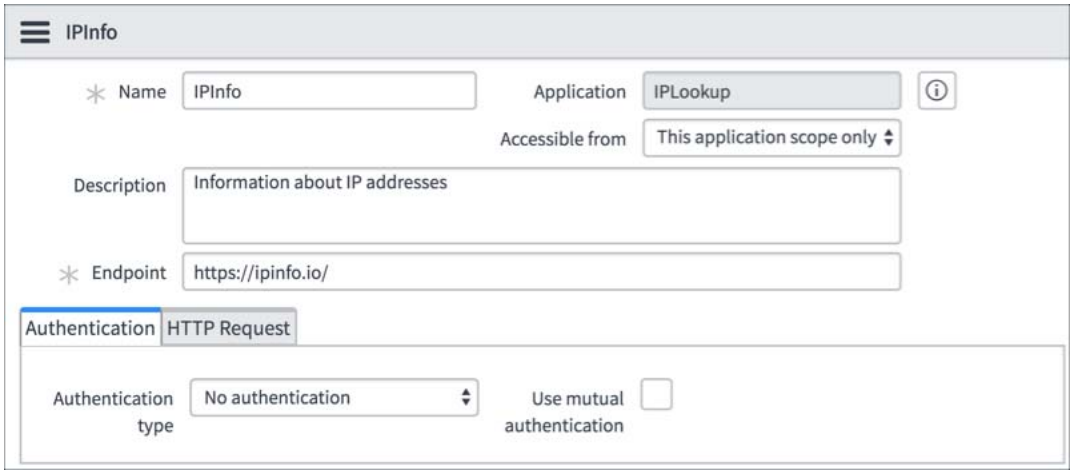
Create the IPLookup Application

1. Use the Application Navigator in the main ServiceNow browser window to open **System Applications > Studio**.

2. Click the **Create Application** button.
3. Click the **Create** button for the **Start from scratch** option.
4. Configure the new application:
   Name: **IPLookup**
   Scope: **this value is automatically populated**
5. Click the **Create** button.
6. When prompted to confirm application creation, click the **OK** button.
7. When the application creation is completed, click the **Back to list** button.
8. Click the link to the **IPLookup** application to open it for editing.

Create an Outbound REST Message

1. Create a REST Message.
   a. In Studio, click the **Create Application File** button.
   b. In the Filter... field enter the text **REST** OR select **Outbound Integrations** from the categories in the left hand pane.
   c. Select **REST Message** in the middle pane as the file type then click the **Create** button.
2. Configure the REST Message:
   Name: **IPInfo**
   Description: **Information about IP addresses**
   Endpoint: **https://ipinfo.io/** (https://ipinfo.io/)
   Authentication type: **No authentication**

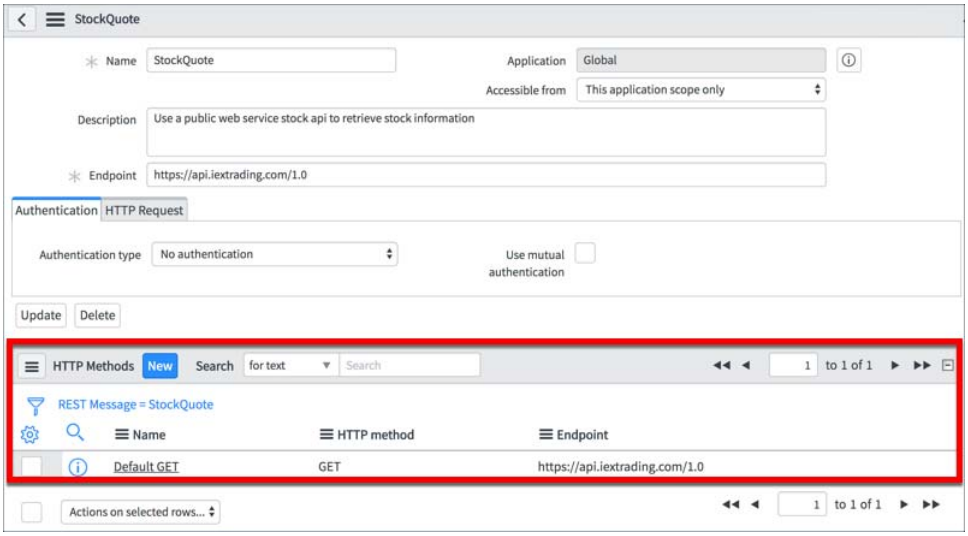


3. Click the **Submit** button to save the REST message.

> NOTE: The ipinfo.io public web service does not require authentication.

1.7 HTTP Methods

HTTP methods define the action to take for a resource such as retrieving information or updating a record. When a new outbound REST Message is saved for the first time, ServiceNow creates an HTTP Method based on information in the outbound REST Message.

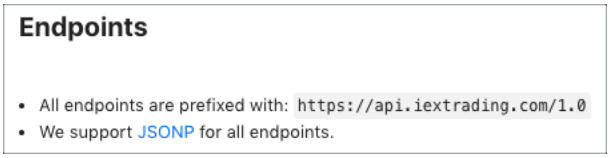


Click the HTTP Method name to open the method for editing or click the **New** button to create an HTTP Method. The available HTTP Methods are:

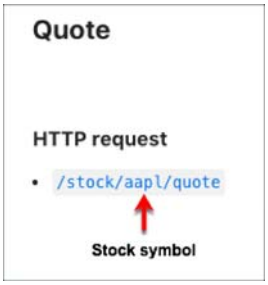
- GET
- POST
- PUT
- PATCH
- DELETE

1.8 Method Endpoint

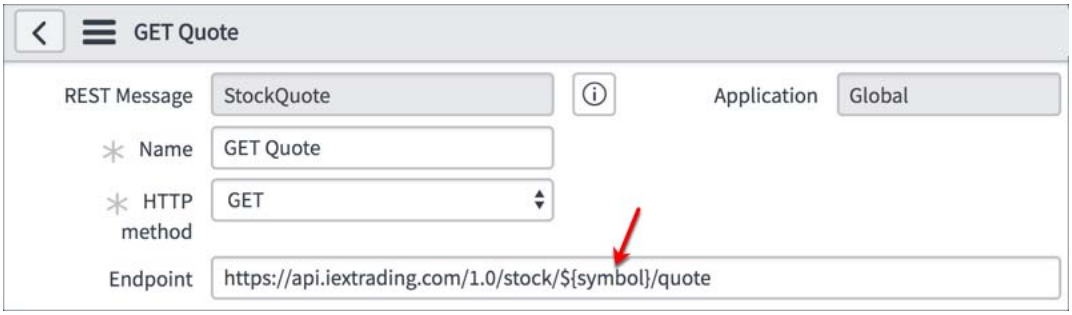
To configure the HTTP method endpoint, refer to the web service provider's API documentation. For example, the iextrading.com API reference provides the base URL for the API:



For the iextrading.com Quote API, the documentation provides the syntax:



To make the Outbound REST Message more useful, instead of hard-coding the company symbol into the endpoint, make the symbol dynamic. Enclose dynamic pieces of endpoints in ${ }.



1.9 Method Authentication and HTTP Request

Authentication

In the default case, HTTP Methods inherit authentication settings from the outbound REST Message (parent). Change the authentication type if it differs from the parent's authentication. The authentication fields for the method are the same as for the outbound REST Message.

HTTP Request - MID Server

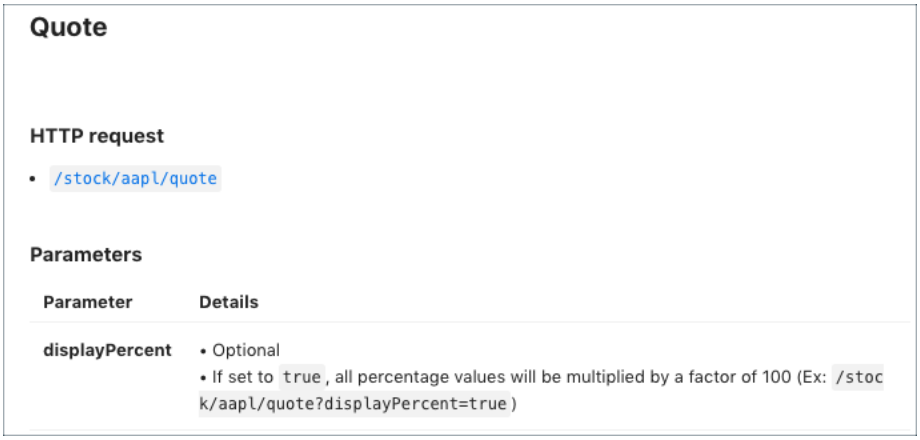
If the web service to be consumed is on an internal company network and not accessible using the Internet, use a **MID Server** (https://docs.servicenow.com/bundle/kingston-servicenow-platform/page/product/mid-server/reference/r-MIDServer.html) to connect to the web service. Select the MID server in the **Use MID Server** field.

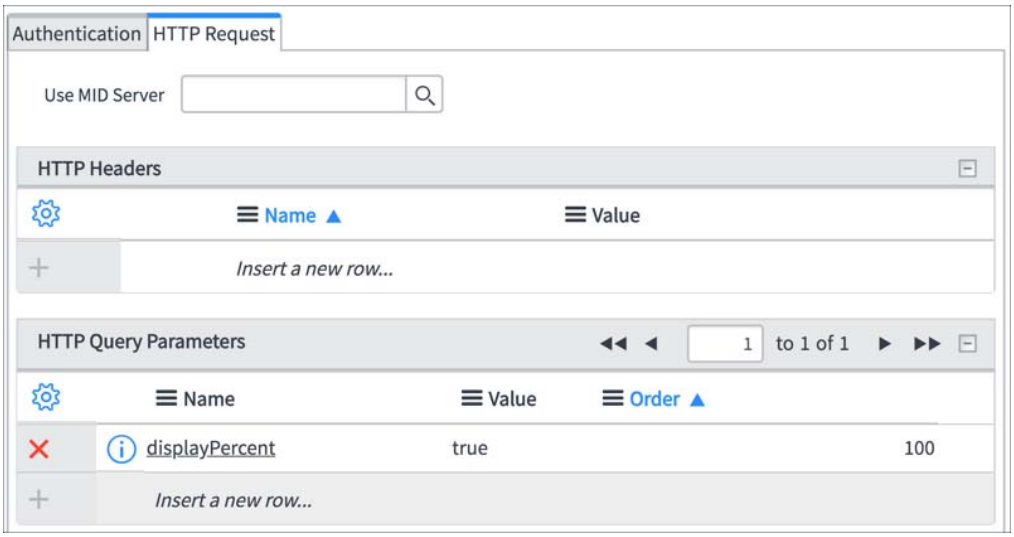
HTTP Request - HTTP Headers

Double-click a row in the HTTP Headers embedded list to define the header Name and Value. The web service provider determines which headers are supported or required. See **List of HTTP Header Fields** (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) for a list of HTTP header fields.

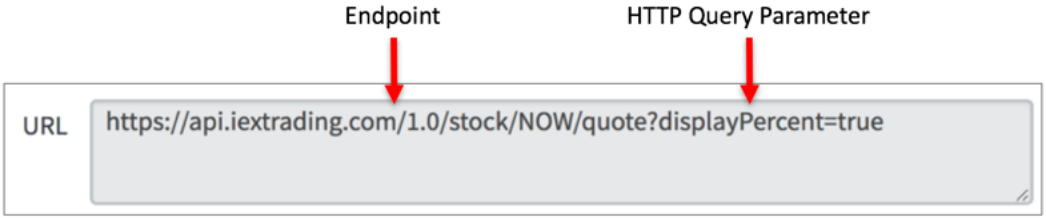
HTTP Request - HTTP Query Parameters

To define an HTTP Query Parameter, double-click the text **Insert a new row...** and provide a Name and a Value for the parameter. Refer to the API's documentation to see which query parameters to define. For example, the iextrading.com site defines an optional query parameter for the Quote API.





HTTP Query Parameters are appended to the endpoint at runtime. The syntax is endpoint + ? + query parameter.



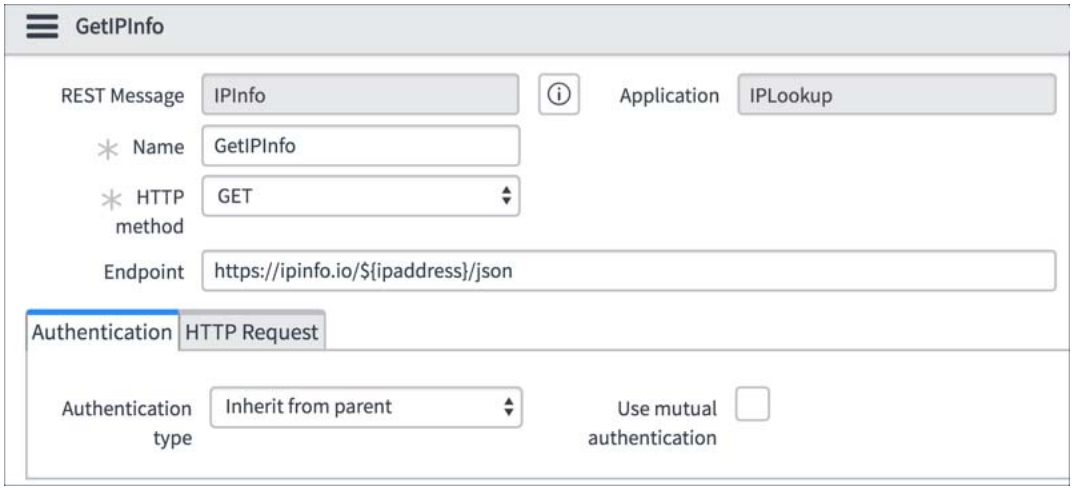
1.10 Exercise: Configure an HTTP Method

In this exercise, you will configure an HTTP Method to get IP address information from the ipinfo.io public web service.

Configure a GET Method

1. When you saved the REST Message in the last Exercise, a related HTTP Method was created. Open it from the related list.

2. Configure the HTTP Method:
   Name: **GetIPInfo**
   HTTP method: **GET**
   Endpoint: **https://ipinfo.io/${ipaddress}/json** (https://ipinfo.io/${ipaddress}/json)
3. Click the **Update** button to save the changes



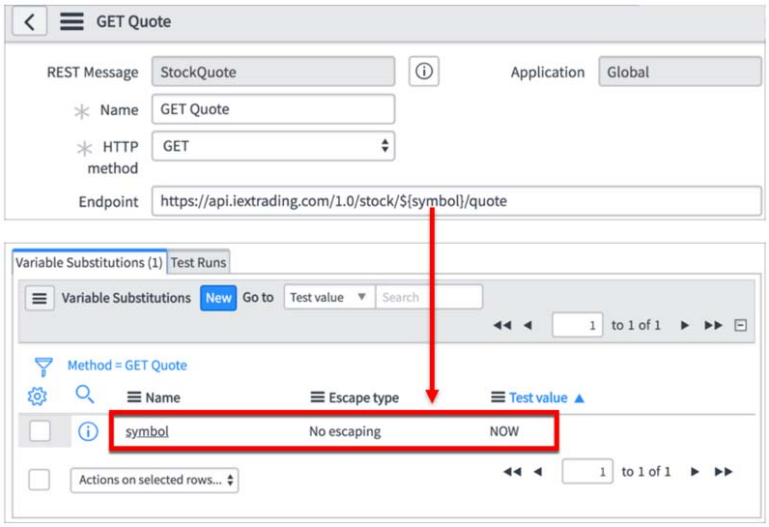
NOTE: The ipinfo.io web service does not require authentication, headers, or query parameters.

1.11 Testing HTTP Methods

After configuring an HTTP method for an outbound REST message, you can test it to ensure the request is valid and the response is what you expected.

Variable Substitutions

Variables defined in the HTTP method must have values in order to test. Click the **Auto-generate variables** related link to automatically add variables to the Variable Substitutions list. For variables not automatically added to the Variables Substitutions list, click the **New** button to manually define variables.

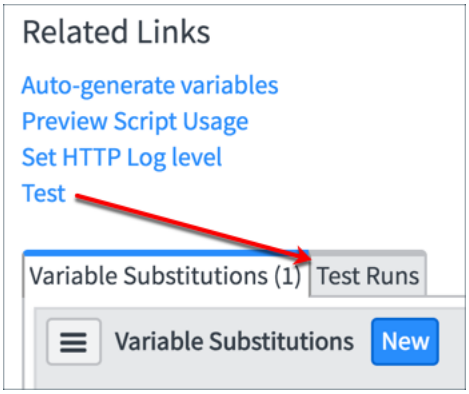


- **Name**: Name of the variable. Must be an exact match to the variable name in the HTTP method.
- **Escape type**: If the data you are passing contains special characters like | (pipe character), set this value to Escape XML.

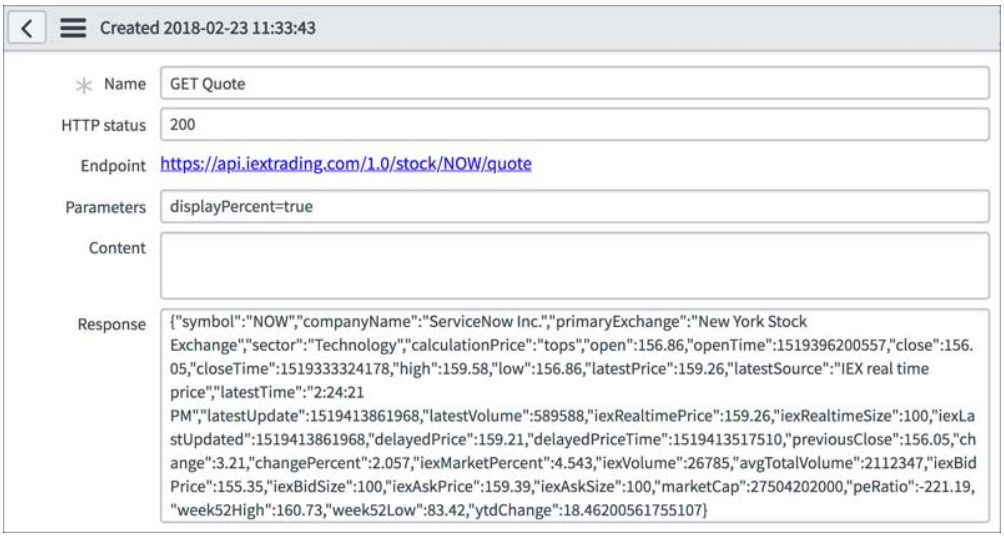
- **Test value**: Value to use for the variable when testing.

Running the Test

To test the HTTP method, click the **Test** related link.



The test results are available in the Test Runs related list.



Examine the response body to make sure the expected data was received.

> NOTE: The HTTP Status indicates the state of the transaction and is **not** an indication that the desired response was received.

1.12 Exercise: Test the HTTP Method

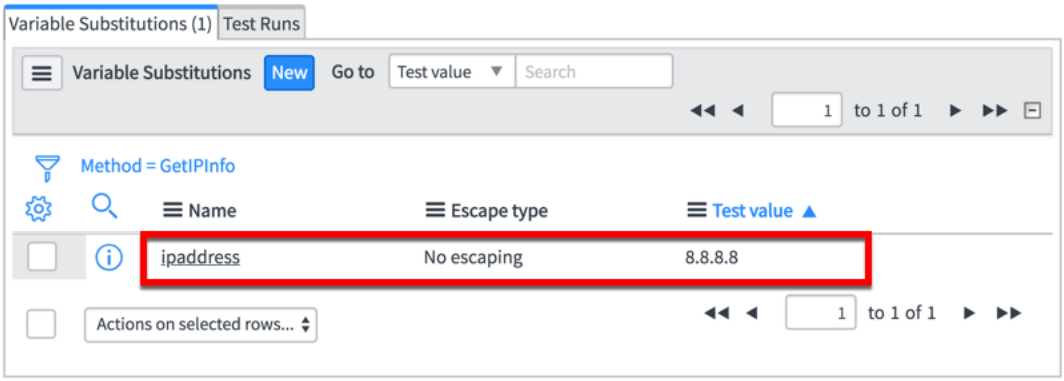
In this exercise, you will test the GetIPInfo HTTP method and will examine the test results.

Preparation

Use the strategy of your choice to determine your IP address. If you are not sure how to find your IP address, you might try **WhatIsMyIPAddress** (https://whatismyipaddress.com/) or **IPChicken** (http://ipchicken.com/). Make a note of your IP address.

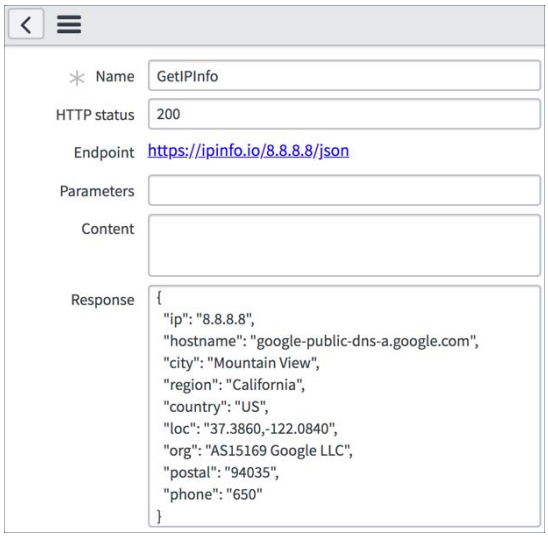
Create Variable Substitution

1. If not still open from the last Exercise, open the **GetIPInfo HTTP method** for editing.
2. Examine the Endpoint field to remind yourself which variable is in the Endpoint URL.
3. Scroll to the **Variable Substitutions** related list (tab) and notice there are no variable substitutions defined.
4. Click the **Auto-generate variables** Related Link.
5. Add a value to the auto-created ipaddress variable. If the ipaddress variable was not auto-created, click the **New** button in the Variable Substitutions section (tab) and add it manually.
    a. Double-click in the **Test value** column.
    b. Enter **your IP address** then click the **OK** button ( ).



Execute the Test and Examine the Results

1. If you had to create the ipaddress Variable Substitution manually, you have to reload the GetIPInfo method form before testing. To execute the test, click the **Test** Related Link.
2. When the test completes examine the results screen. Your test results should look something like this but with values for your IP address:



3. Is the HTTP status value **200**?
4. What information do you see in the Response?

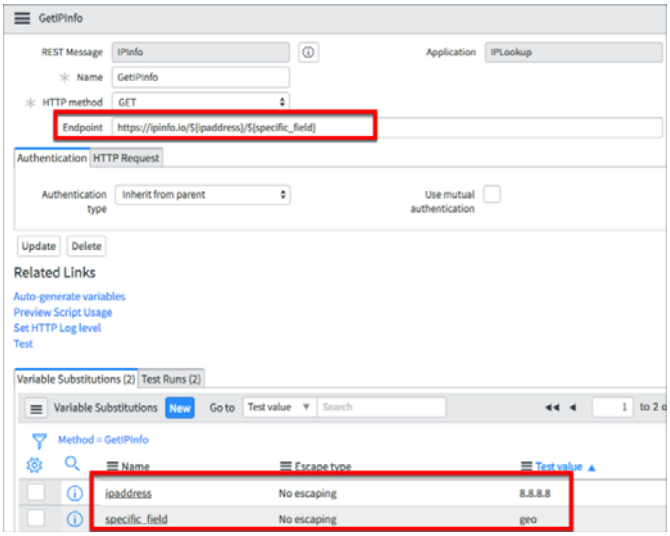
Challenge - Request Specific Field Data

Your Challenge is to modify the GetIPInfo HTTP method to allow the method to request a specific field's data from the ipinfo.io API. See the Specific Fields section of the **ipinfo.io API developer documentation** (https://ipinfo.io/developers). The GetIPInfo HTTP method must accept a value of org, city, geo, or json. Test your solution using the Test Related Link.

If you get stuck, you can find a solution to the Challenge in the Answers section at the bottom of this page.

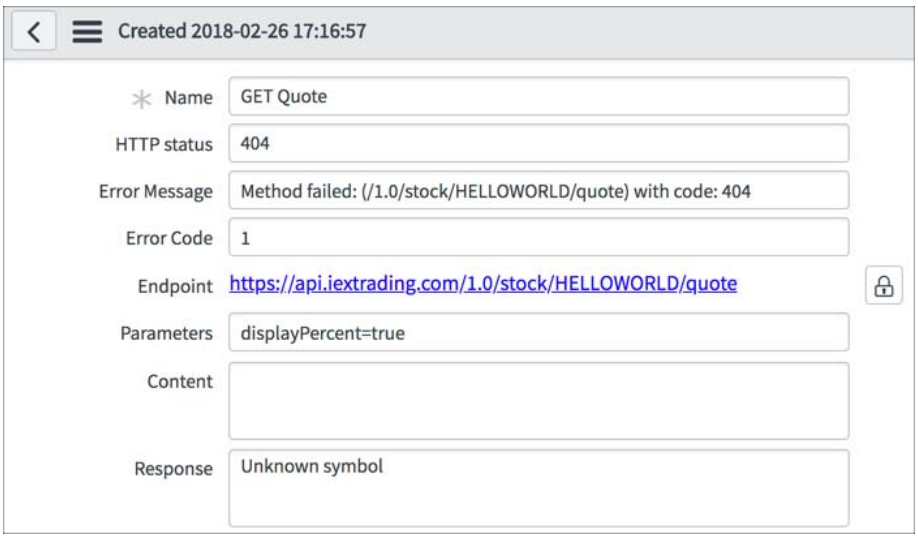
Answers

Challenge:



1.13 Debugging HTTP Methods

The first place to look when debugging REST HTTP methods is the test results page. In this example, the HTTP status returned a 404 which indicates endpoint was not found. The Response gives further information about the source of the error.

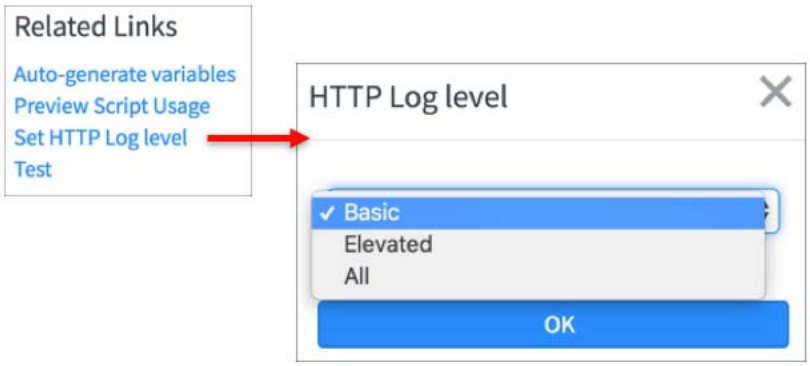


Outbound HTTP Requests System Log

If more in-depth debugging is required, use the Outbound HTTP Requests system log. Begin by setting the verbosity of the logging. Click the **Set HTTP Log level** Related Link in the HTTP method definition to specify a logging level.

- Basic: least amount of debugging information

- Elevated: more debugging information
- All: most debugging information

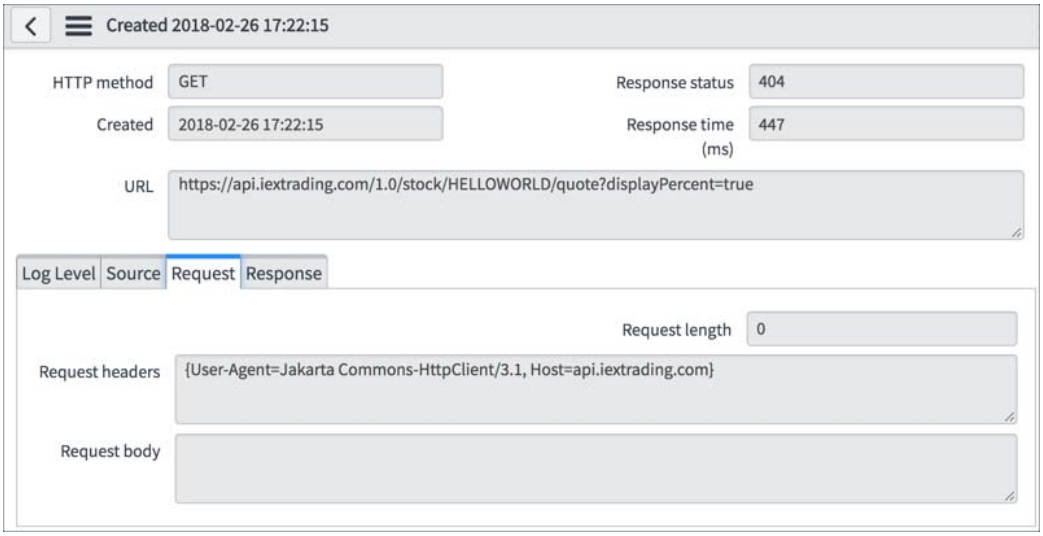


After setting the log level, test again to generate the log.

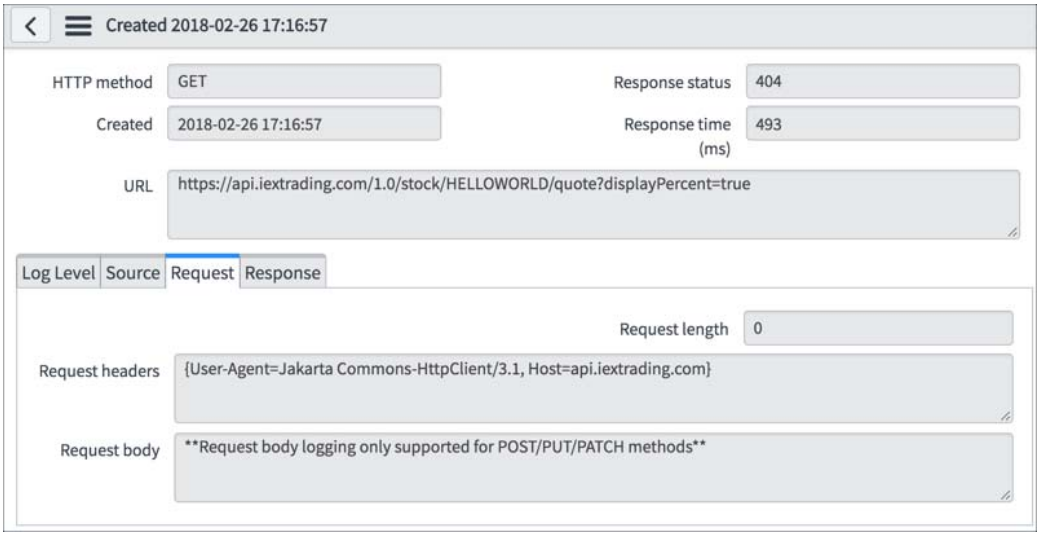
Use the Application Navigator to open **System Logs > Outbound HTTP Requests**.

Basic

Elevated



All



Although the example shows the Request tab, depending on what is being debugged, the other tabs may be more useful.

> DEVELOPER TIP: Log levels can also be set using the **System Web Services > Outbound > HTTP Log Levels** module.

1.14 Exercise: Outbound HTTP Requests System Log

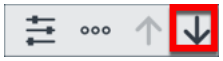
In this exercise, you will set the log level for the GetIPInfo HTTP method and will examine the Outbound HTTP Requests System Log.

Set Logging Level and Generate Log

1. If not still open from the last Exercise, open the **GetIPInfo HTTP** method for editing.
2. Set the HTTP Log level value to Basic.
    a. Click the **Set HTTP Log level** Related Link.
    b. If the HTTP Log level value is not **Basic**, set the value to **Basic**.
    c. Click the **OK** button.
3. Click the **Test** Related Link to execute the test and generate the Basic log.
4. Return to the GetIPInfo HTTP method and repeat the steps to generate logs for the **Elevated** and **All** log levels.

Examine the Outbound HTTP Requests Log

1. In the main ServiceNow browser window (not Studio), use the Application Navigator to open **System Logs > Outbound HTTP Requests**.
2. Open the first log record in the list.
3. Examine the log.
    a. Examine **Log Level** section (tab) to determine the log level of the record.
    b. Switch to the **Request** section (tab) and examine the contents.
    c. Switch to the **Response** section (tab) and examine the contents.
4. Use the **Next Record** icon to move to the next record in the log list.



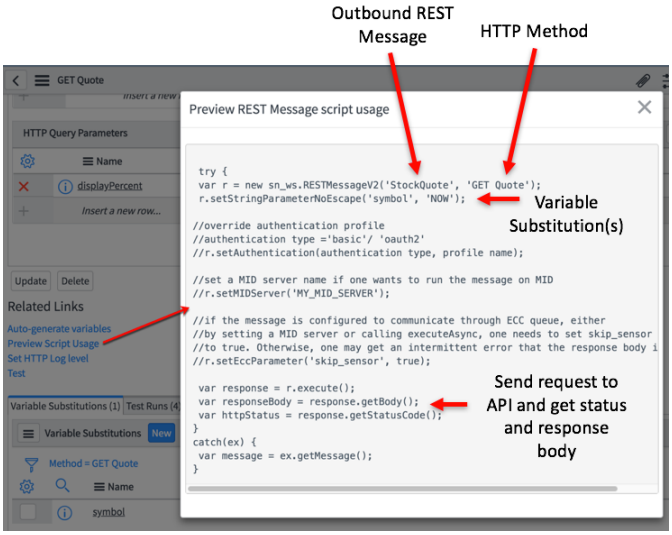
5. Determine the log level for the record and examine the sections (tabs) to see what information they contain.

6. Repeat this process for the last record in the log.

1.15 Preview Script Usage

In most cases, it is impractical to use the Test Related link in the outbound REST Message methods whenever you want to invoke the API. After configuring an HTTP method and verifying it works as expected, the final development step is execute the method from a server-side script.

The Preview Script Usage Related Link generates a server-side JavaScript code stub for invoking the API.



The **RESTMessageV2 API** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_RESTMessageV2API) sends outbound REST messages using JavaScript.

In the sample script, the .setStringParameterNoEscape method is passed the parameter name and a hard-coded value. When implementing the script, the value for the parameters is typically set dynamically. For example:

```
r.setStringParameterNoEscape('symbol',current.company_ticker);
```

You may have noticed the syntax sn_ws.RESTMessageV2() in the sample script. If you have written server-side scripts in ServiceNow before, you may have written a script which instantiates the GlideRecord class.

```
var myObj = new GlideRecord('table_name');
```

The GlideRecord class is part of the ServiceNow global namespace and does not need to have its namespace qualified in the script. The RESTMessageV2() API is part of the sn_ws (Web Service) namespace. Although its methods are accessible to all scopes, attempting to instantiate RESTMessageV2 without prepending the namespace would return an error because the API would not be found in the global namespace. Prepending the namespace tells ServiceNow where to find the API.

1.16 Parsing Data from the Response

The sample script from Preview Script Usage has code to get the body of the response and the HTTP status code.

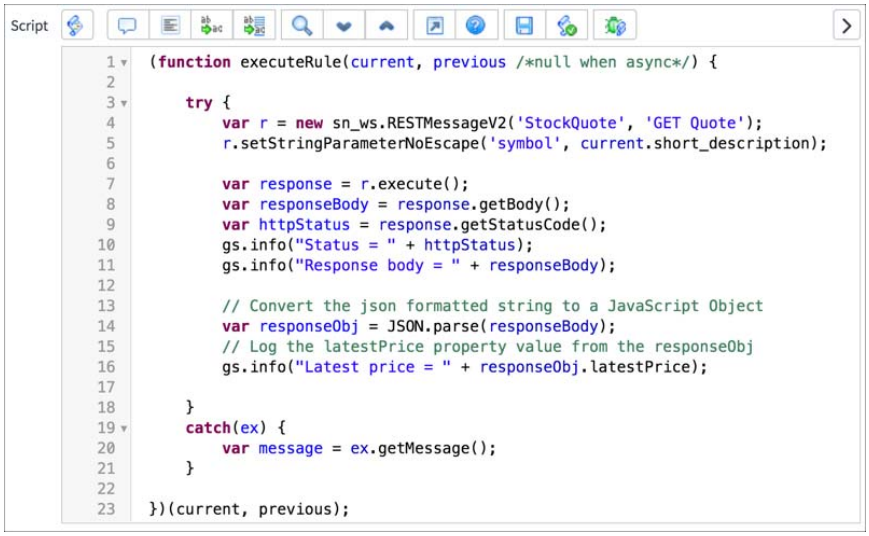
```
var response = r.execute();
    var responseBody = response.getBody();
    var httpStatus = response.getStatusCode();
```

There is no sample code in the Preview Script Usage script for extracting data from the response. The responseBody variable contains the response as a string. You could use JavaScript string methods such as substring(), substr(), or indexOf() to extract data from the responseBody variable.

JSON

When working with a json response, use the **JSON API** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_JSONAPI) which converts a JSON formatted string to a JavaScript object. When working in the Global scope, the following methods are available:

- decode (deprecated)
- encode (deprecated)
- parse
- stringify



When working in privately-scoped applications, the following methods are available:

- parse
- stringify

See the **JSON API documentation** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_JSONAPI) for additional examples of how to use these methods.

XML

When working with an XML response in privately-scoped applications, use the **XMLDocument2 API** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_XMLDocument2ScopedAPI) to extract data from the response body.

```
var response = r.execute();
var responseBody = response.getBody();
var httpStatus = response.getStatusCode();
gs.info("Status = " + httpStatus);
gs.info("Response body = " + responseBody);

// IMPORTANT:  This example shows what the script MIGHT look like if
// responseBody were an XML formatted string.

// Create an XMLDocument2 object
var xmlDoc = new XMLDocument2();
// Read the responseBody XML string into the XMLDocument2
xmlDoc.parseXML(responseBody);
// Log the value of the latest price XML node
gs.info(xmlDoc.getNodeText("//latestPrice"));
```

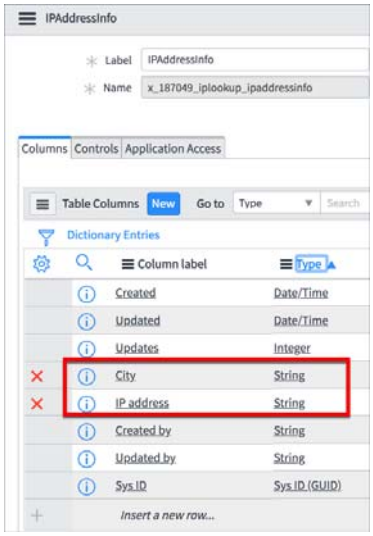
NOTE: This XML example is for demonstration purposes. It is not based on real data from the iextrading.com API.

1.17 Exercise: Write a Script to Invoke the REST Message

In this exercise, you will write a script to invoke the IPInfo Outbound REST Message based on the IP Address value in a record. You will update the City field in the record by parsing data from the web service response.

Preparation

1. Add a table to the IPLookup application.
    a. If not still open from a previous Exercise, open the IPLookup application in Studio for editing.
    b. Click the **Create Application File** button.
    c. In the Filter... field enter the text **Table** OR select **Data Model** from the categories in the left hand pane.
    d. Select **Table** in the middle pane as the file type then click the **Create** button.
2. Configure the table:
    Label: **IPAddressInfo**
    Name: **this value is automatically populated**
3. Click the **Submit** button to save the table.
4. Add a field to the IPAddressInfo table.
    a. Scroll to the Columns section (tab) and double-click on **Insert a new row...**.
    b. Configure the new field.
        Column label: **IP address**
        Type: **String**
5. Add another field to the IPAddressInfo table:
    Column label: **City**
    Type: **String**
6. Click the **Update** button.



Preview Script Usage

1. If not still open, open the GetIPInfo HTTP method for editing.
2. Scroll to the Related Links and click the **Preview Script Usage** link.
3. Copy the Preview REST Message script usage script to the clipboard.

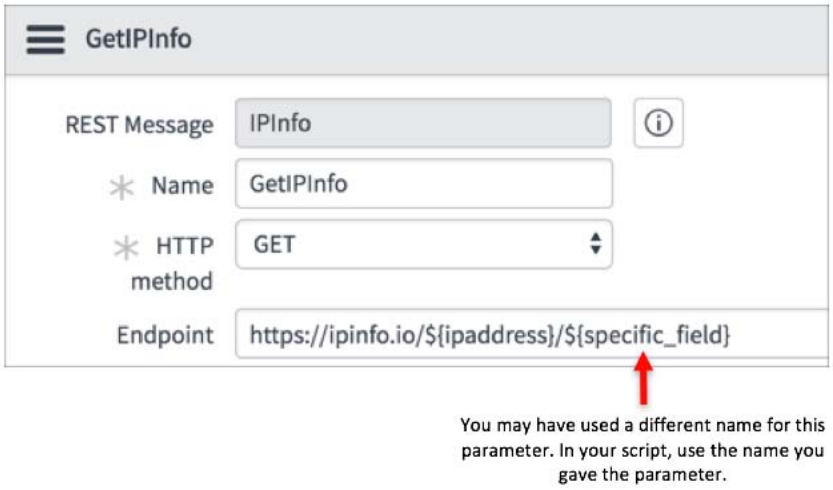
Create a Business Rule

1. Create a Business Rule in the IPLookup application.
    a. Click the **Create Application File** button.
    b. In the Filter... field enter the text **Business** OR select **Server Development** from the categories in the left hand pane.
    c. Select **Business Rule** in the middle pane as the file type then click the **Create** button.
2. Configure the Business Rule:
    Name: **Populate IP Address City**
    Table: **IPAddressInfo**
    Advanced: **Selected (checked)**
3. Configure the Populate IP Address City Business Rule's When to Run section (tab).
    When: **async**
    Insert: **Selected (checked)**
    Update: **Selected (checked)**
4. Click the **Submit** button to save the Business Rule.

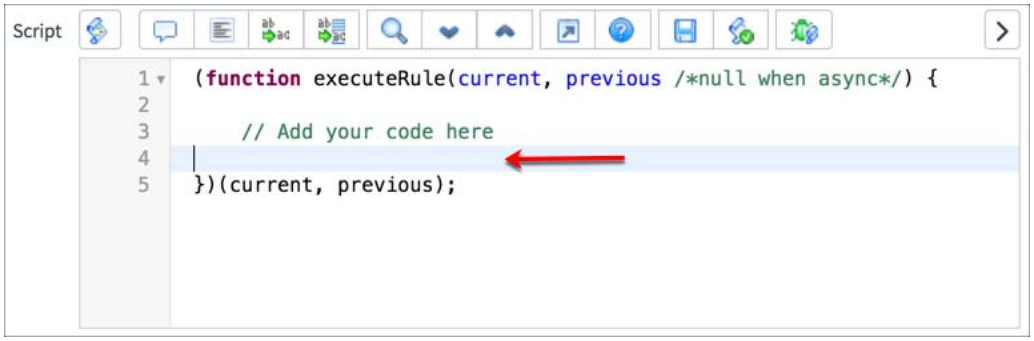
> DEVELOPER TIP: Business Rules which invoke a web service **must be async**.

Add a Script to the Business Rule

> NOTE: The second parameter in the Endpoint is referred to as "specific_field" in this Exercise.



1. Switch to the Advanced section (tab).
2. Paste the code stub you copied into the Business rule after the comment which says // Add your code here.



3. In the script, set the specific_field parameter value to **'geo'**.

4. In the script, set the ipaddress parameter value to **current.ip_address**.

```
r.setStringParameterNoEscape('specific_field', 'geo');
r.setStringParameterNoEscape('ipaddress', current.ip_address);
```

5. Add logic to populate the value of the City field on the IPAddressInfo record from the city property in the response and update the record.

```
// Convert the json formatted string to a JavaScript Object
var responseObj = JSON.parse(responseBody);
// Set the value of the City field on the IPAddressInfo record
current.city = responseObj.city;
gs.info("City = " + current.city);
current.update();
```

6. The code stub assumes the error messages have been internationalized which we have not done. Modify the catch() logic to report the errors caught from the try:

```
catch(err) {
    gs.info("Uncaught error: " + err);
}
```

7. Click the **Update** button.

Test the Script

1. Switch to the main ServiceNow browser window (not Studio) and reload the browser window. This will load the module for the IPAddressInfo table into the Application Navigator.
2. Use the Application Navigator to open **IPAddressInfo > IPAddressInfos**.
3. Click the **New** button.
4. Enter your IP address in the IPAddress field.
5. To save the record and remain on the form page, click the **Additional Actions** menu ( ) and select the **Save** menu item.
6. Allow time for the async Business Rule to execute. The City field should be populated after the Business Rule executes.

> QUESTION: What does the circle to the left of the City field indicate? If you are not sure, you can find the answer in the Answers section at the bottom of this page.

7. If the City field does not have a value, use the Application Navigator to open the **System Logs > System Log > All** module to debug the problem with the Business Rule.

Answers

> **Question**: What does the circle to the left of the City field indicate?
> **Answer**: The blue circle indicates the form field was updated by a script or another user while the current user was viewing the form.

1.18 Scripting REST Messages

In the typical outbound integration case, developers define an Outbound REST Message as demonstrated in this module. The benefit of Outbound REST Messages is that they allow an API interaction to be configured, tested, and debugged using a form-based UI.

Some developers prefer to script the entire integration without creating an Outbound REST Message. This script defines an interaction with the iextrading.com Stock Quote API *without* using an Outbound REST Message:

```
// Create an empty RESTMessageV2 object
    var getQuote = new sn_ws.RESTMessageV2();
    // Set the endpoint
    getQuote.setEndpoint('https://api.iextrading.com/1.0/stock/${symbol}/quote');
    // Set the HTTP method (get, post, put, patch, delete)
    getQuote.setHttpMethod('get');
    // Set HTTP Query Parameters
    getQuote.setQueryParameter('displayPercent','true');
    // Set values for endpoint variables
    getQuote.setStringParameterNoEscape('symbol', current.short_description);
```

There are other useful methods for scripting REST interactions in the RESTMessageV2 API such as:

- setRequestHeader()
- setLogLevel()
- setStringParameter()
- setRequestorProfile()
- setMutualAuth()

See the **RESTMessageV2 API documentation** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_RESTMessageV2API) for the complete list of methods.

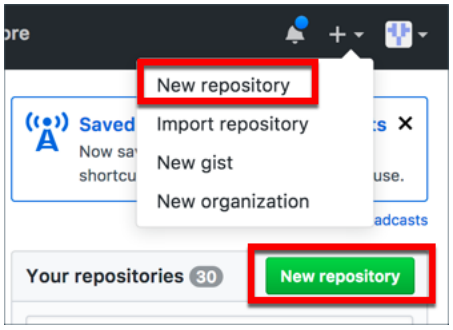
1.19 Exercise: Save the IPLookup Application (Optional)

In this exercise, you will save the work you have done in this module to your GitHub repository.

> **NOTE**: For more details about connecting to GitHub, visit the **GitHub and ServiceNow section** (https://developer.servicenow.com/app.do#!/program/faq?q=github_and_servicenow) of the Developer Program FAQ.

Log in to GitHub and Create a Repository

1. Open the **github.com** (https://github.com/) web site.
2. If you have a GitHub account, log in. If you do not have an account, create an account and log in.
3. Create a new repository using the **Add** menu or clicking the **New repository** button.



4. Configure the new repository:

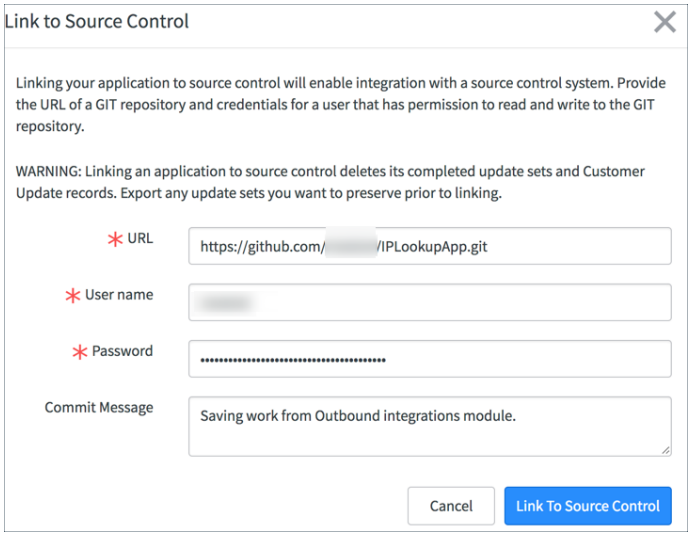
   Repository name: **IPLookupApp**
   Description: **IPLookup application from the ServiceNow Outbound REST integrations module**

5. Click the **Create repository** button.
6. After the IPLookupApp repository is created, click the **Copy to clipboard** button to copy the https URL.

Connect Studio to the IPLookupApp Repository

1. If the IPLookup application is not open in Studio from the last exercise, open it now.
   a. In the main ServiceNow browser window use the Application Navigator to open **System Applications > Studio**.
   b. In the Load Application dialog, click the **IPLookup** application.
2. Open the **Source Control** menu and select the **Link to Source Control** menu item.
3. Configure the connection the IPLookupApp GitHub repository:

   URL: **URL you copied after creating the IPLookupApp repository**
   User name: **Your github.com user**
   Password: **Your github.com password**
   Commit Message: **Commit message of your choice**



4. Click the **Link To Source Control** button.

5. When the Link to Source Control dialog reports success, click the **Close Dialog** button.

6. The Studio footer shows that Studio is connected to the repository's master branch.

1.20 Outbound Integrations Module Recap

Core concepts:

- Use outbound REST Messages to define, test, and debug interactions with third-party web service providers
- HTTP methods include:
    - Endpoint
    - HTTP verb (get, post, put, patch, delete)
    - HTTP Headers
    - MID server selection
    - HTTP Query Parameters
    - Authentication
- Create Variable Substitutions before using the Test Related Link

- Examine test results carefully
    - HTTP Status of 200 does not mean the expected data was returned
    - Examine the Response to see if the expected data and format were received

- Use the **Set HTTP Log Level** Related Link to change the log level
    - Choose from Basic, Elevated, or All
    - The default log level is basic
    - Use the Outbound HTTP Requests log to view debugging information

- The Preview Script Usage Related Link creates a JavaScript code stub for interacting with the web service provider from a server-side script
    - When possible make the setStringParameter() variables values dynamic rather than hard-coded
    - Add script logic to extract information of interest from the REST response
    - JavaScript string methods
    - JSON.parse
    - XMLDocument2 methods
    - Might need to modify the script logic for the catch block

- Use the RESTMessageV2 API to script a REST interaction with a third-party web service provider without first defining an Outbound REST Message

2. Inbound REST Integrations

In this module you will practice building and testing requests to the ServiceNow APIs using the REST API Explorer.

2.1 Inbound Integrations in ServiceNow Objectives

In this module you will learn to:

- Use the REST API Explorer and the Table API to interact with records from ServiceNow tables
    - Path parameters
    - Query parameters
    - Headers
    - HTTP Status
    - Response Body

- Create a ServiceNow user for inbound REST requests
- Create cross-origin resource sharing (CORS) rules to select which HTTP methods are allowed from a resource
- Disable web service access to tables
- Create code samples to use in third-party applications which integrate into ServiceNow

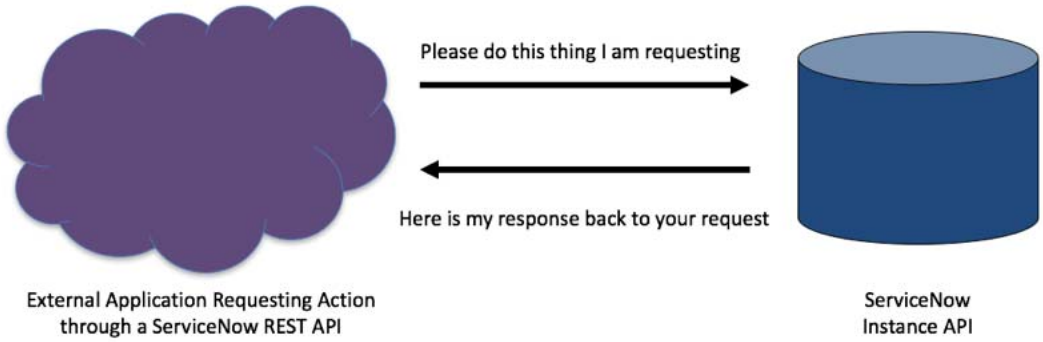
2.2 ServiceNow as a Web Service Provider

Web services make it possible for applications to connect to other software applications over a network allowing an exchange of information between the provider (server) and client (consumer).

A web service consumer (client) requests information from a web service provider (server). A web service provider processes the request and returns a status code and a response body. When the response body is returned, the web service consumer extracts information from the response body and takes action on the extracted data.

In an inbound request, a third-party application requests the ServiceNow APIs take action. Example ServiceNow APIs include:

- Table API: Create, read, update, and delete records from a table
- Attachment API: Upload and query file attachments
- Email API: Send and receive email messages using REST



The complete set of **ServiceNow REST APIs** (https://docs.servicenow.com/bundle/kingston-application-development/page/build/applications/concept/api-rest.html) is documented on the docs.servicenow.com site. Use the **Next Topic** and **Previous Topic** buttons to navigate through the APIs.

The REST API capability is active by default in all instances.

2.3 Introduction to the REST API Explorer

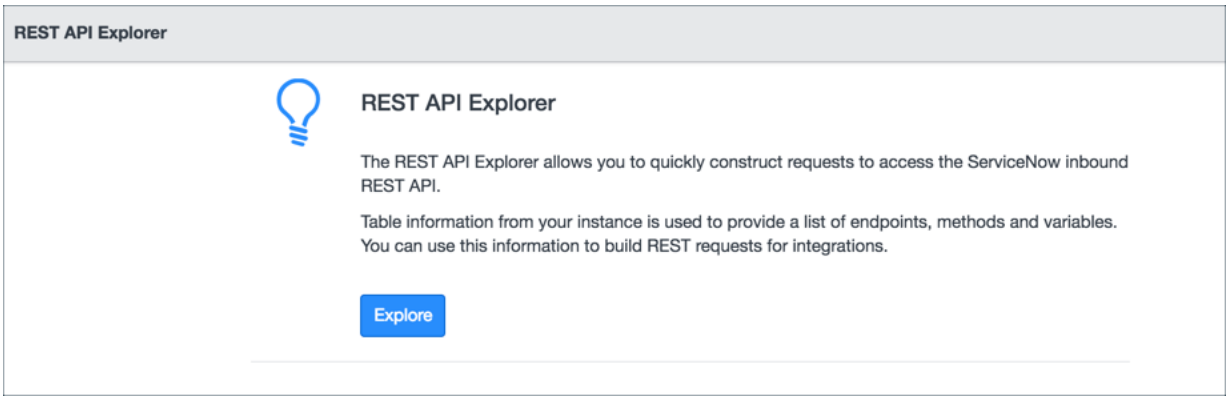
To integrate to any web service, developers need to know:

- Endpoints
- Methods
- Variables

ServiceNow's REST API Explorer is an application for constructing and testing API requests to a ServiceNow instance.

The REST API Explorer is available to users with the rest_api_explorer role. To open the REST API Explorer, use the Application Navigator to open **REST > REST API Explorer**.

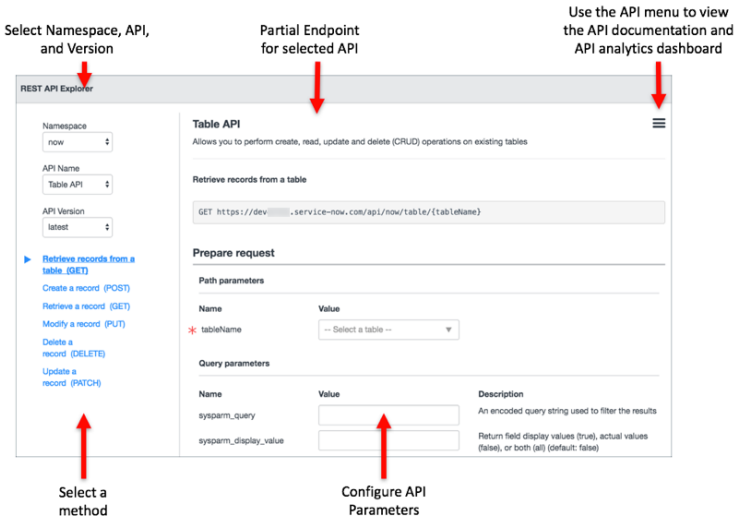
The *first time* the REST API Explorer explorer launches for a user, ServiceNow displays a welcome screen.



The REST API Explorer consists of:

- A pane for selecting the Namespace, API, API version, and REST method

- A pane for viewing and configuring the endpoint
- A menu for accessing documentation for the selected API and an API analytics dashboard
- A section for testing the endpoint (not shown in the screenshot)

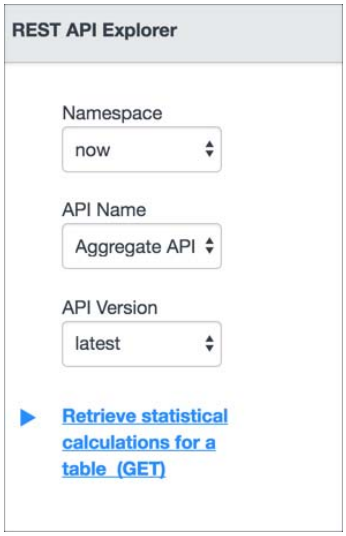


IMPORTANT: The REST API Explorer interacts directly with the tables on your instance. The REST API Explorer can delete, update, and insert records. **We recommend practicing with the REST API Explorer on a non-production instance.** *Use the REST API Explorer only with caution on production instances.*

2.4 Selecting an API

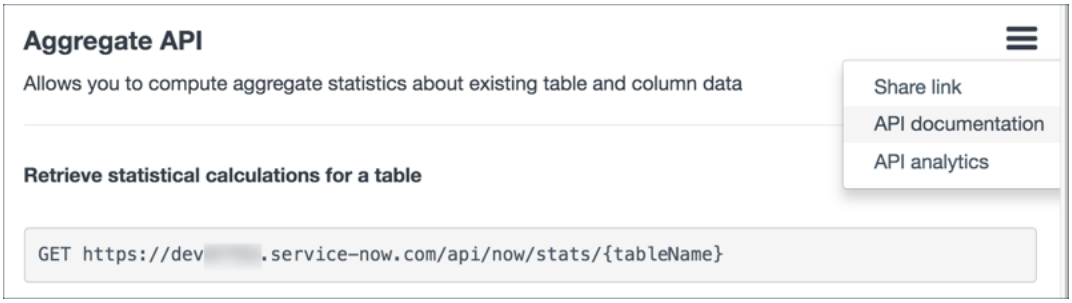
The fields in the **Prepare request** section of the REST API Explorer form are determined by which Namespace, API Name, API Version, and REST method are selected.

- **Namespace**: Web service scope
  - **Global**: Globally scoped APIs
  - **now**: REST APIs provided by ServiceNow
  - **private_scope_name**: APIs (scripted web services) in privately-scoped applications
- **API Name**: Select an API to configure and test in the REST API Explorer
- **API Version**: Select a specific API version or choose latest
- **Method**: Selectable list of available REST methods based on the Namespace, API Name, and API Version. The blue arrowhead indicates the selected method.



For the Aggregate API, only one method is available.

The REST API Explorer displays information about the selection. For more information, select the **API documentation** menu item from the REST API Explorer menu.



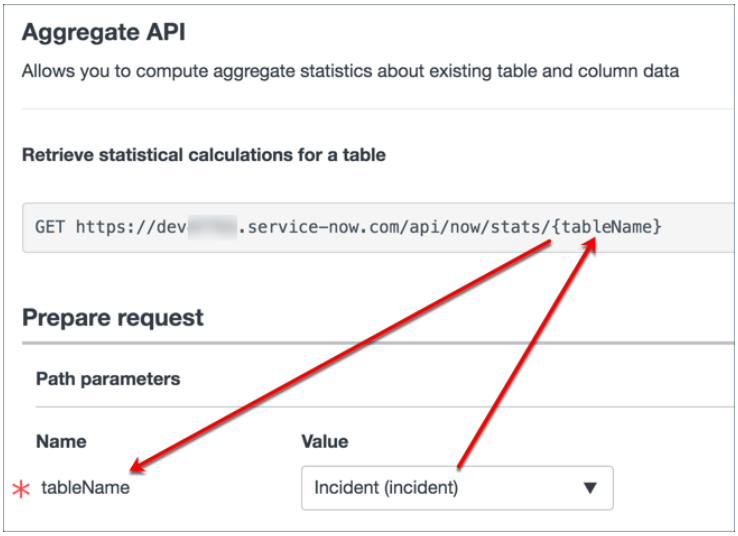
2.5 Request Parameters

Request Parameters consist of:

- Path parameters
- Query parameters
- Request headers

Path Parameters

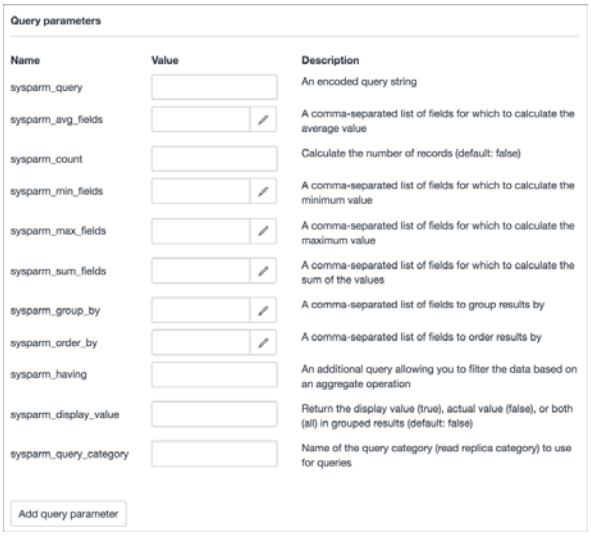
The list of path parameters depends on the endpoint URL. Path parameters are enclosed in curly braces in the endpoint URL. The values set in the path parameter field are substituted into the endpoint URI when a request is sent.



IMPORTANT: If the table referenced by a request is a Database View, GET is the only method which will return results. Database Views are read-only.

Query Parameters

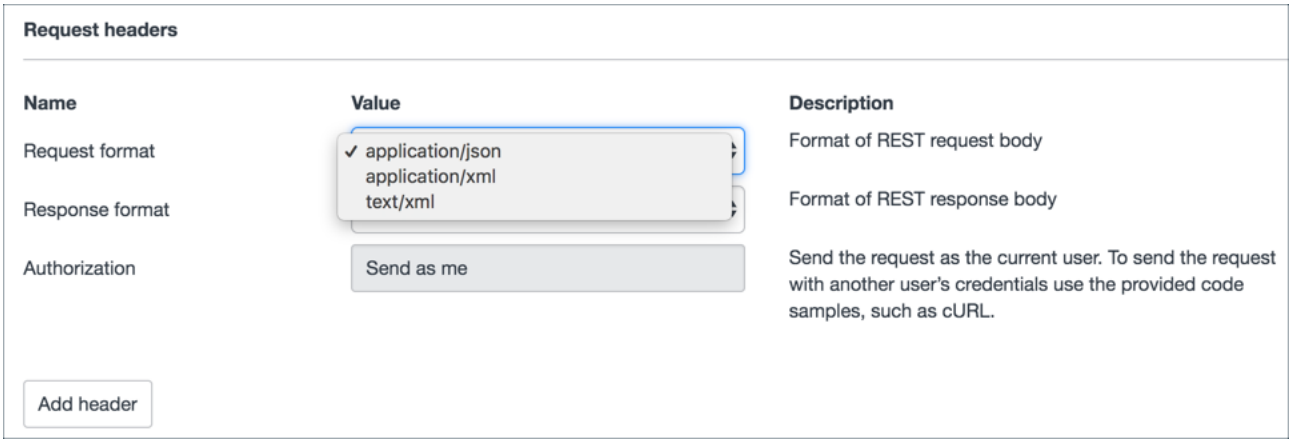
Request parameters are appended to the endpoint URL by the REST API Explorer when the request is sent. The query parameters are specific to the selected API method.



A default set of query parameters are displayed for the API. To add additional query parameters, use the **Add query parameter** button to add a new parameter to the query. For a complete list and detailed description of an API's query parameters, select the **API documentation** menu item from the REST API Explorer menu.

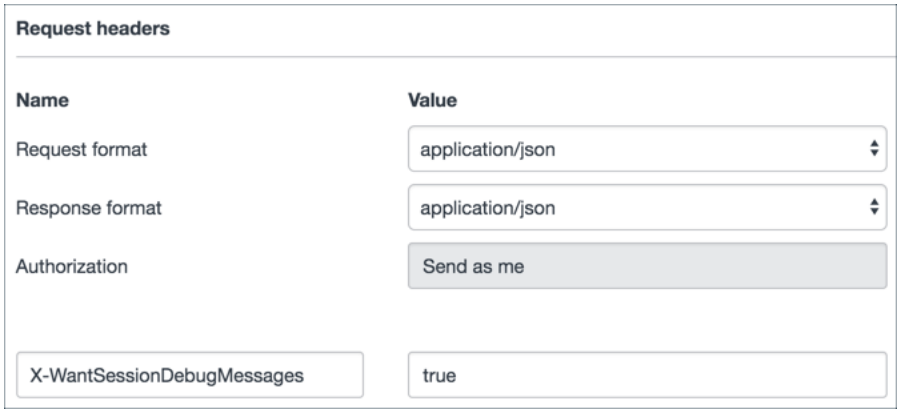
Request Headers

Request headers define the format of the Request and Response.



Use the **Add header** button to add additional headers to the request. For the ServiceNow APIs, the two additional header parameters you may want to add are:

- X-WantSessionNotificationMessages: Set to true to return notifications which have not already been consumed for the existing session.
- X-WantSessionDebugMessages: Enable Session Debug and set the header value to true to return session debug logs.



IMPORTANT: The request headers require you to configure the request as the currently logged in user. The user must have permission to access the records being requested or the request will be denied.

2.6 More About Query Parameters

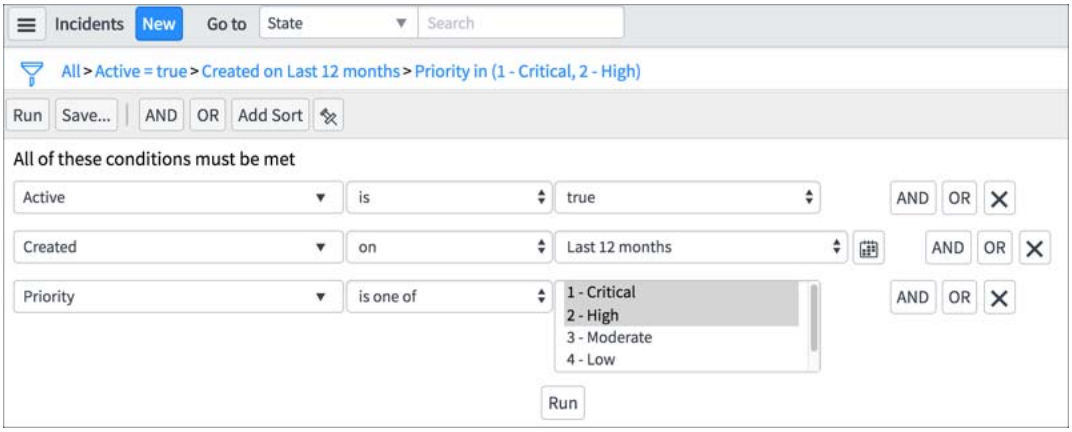
Query parameters may contain different types of values such as:

- Encoded queries
- Field name(s)
- true/false/all

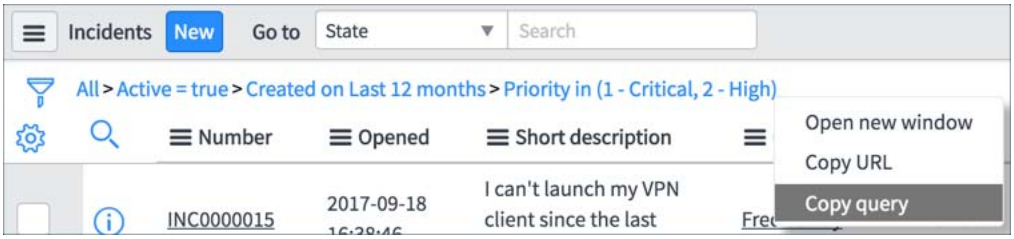
Encoded Queries

Some query parameters, such as sysparm_query and sysparm_having, accept encoded queries as their values. The trick to making this work is to know the encoded query syntax. The syntax is not documented so the best thing to do is let ServiceNow build the encoded query for you. In the main ServiceNow browser window, use the Application Navigator to open the list for the table of interest. If there is no module to open the list, type <table_name>.list in the filter field in the Application Navigator.

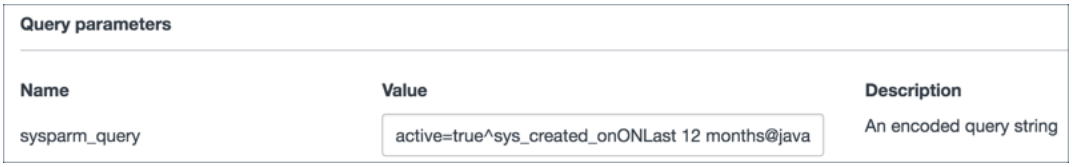
Use the Filter to build the query condition.



Click the Run button to execute the query. Right-click the breadcrumbs and select **Copy query**. Where you click in the breadcrumbs matters. The copied query includes the condition you right-click on and all conditions to the left. To copy the entire query, right-click on the condition farthest to the right.



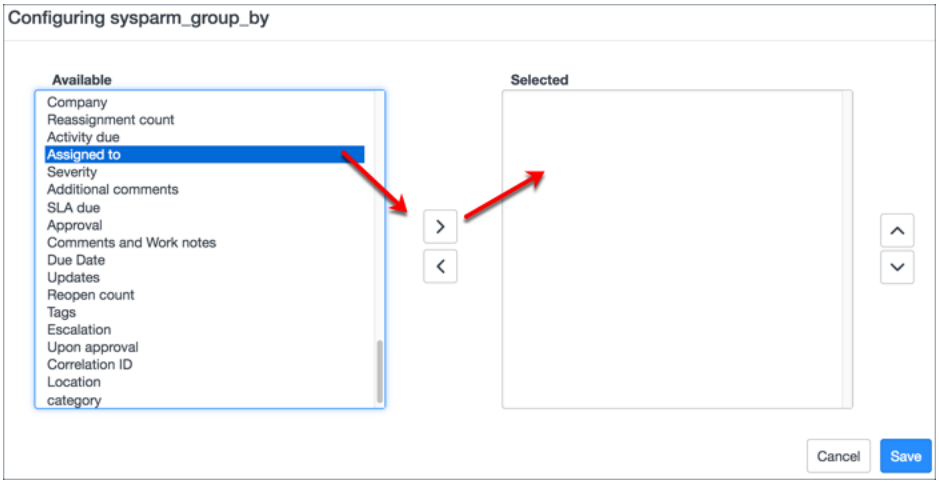
Return to the REST API Explorer and paste the encoded query into the query parameter.



Field Names

Some query parameters, such as sysparm_group_by, accept a field name or a comma separated list of field names. Developers must pass the field *name* and not the field *label*.

To select fields from a slushbucket, click the **Edit** button () on the field.

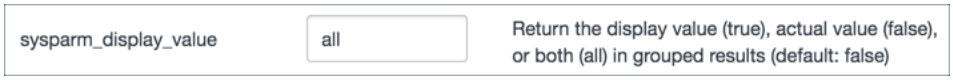


Dot-walking is allowed in field names, for example, caller_id.title.

true/false/all

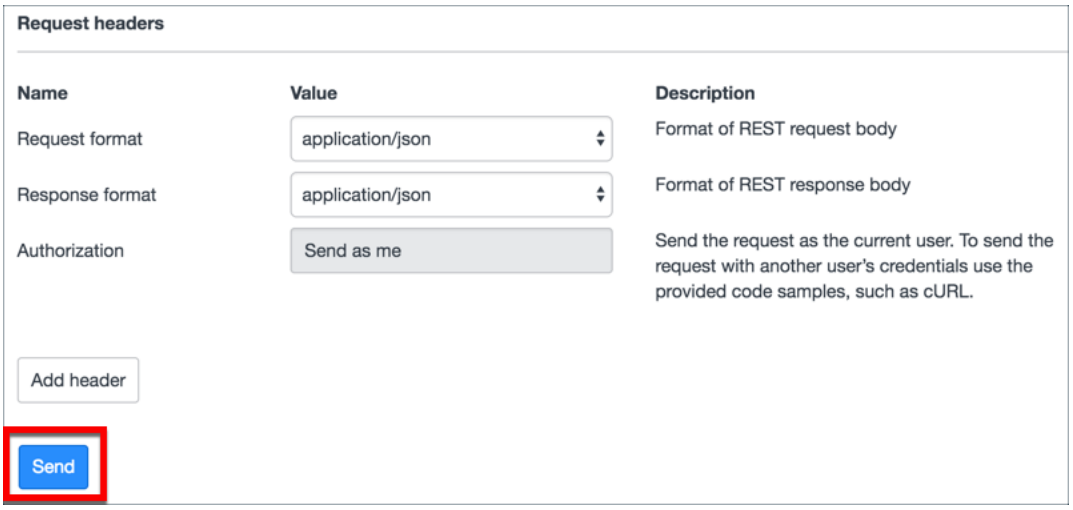
When a REST call returns a reference field from a table, developers can select the format for the returned value.

- Set the value to **true** to return the display value
- Set the value to **false** to return the sys_id
- Set the value to **all** to return both the display value and the sys_id



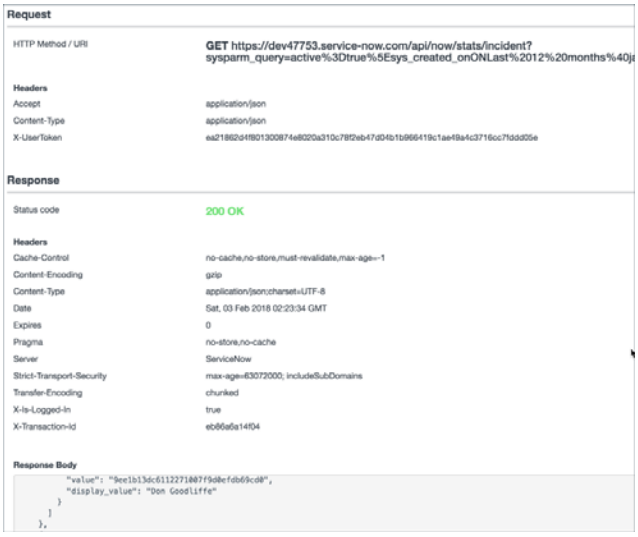
2.7 Testing

After configuring the REST method, click the **Send** button to send the request to the API.



The REST API Explorer responds as if the request had come from a third party application:

- Request
- Response
- Response body

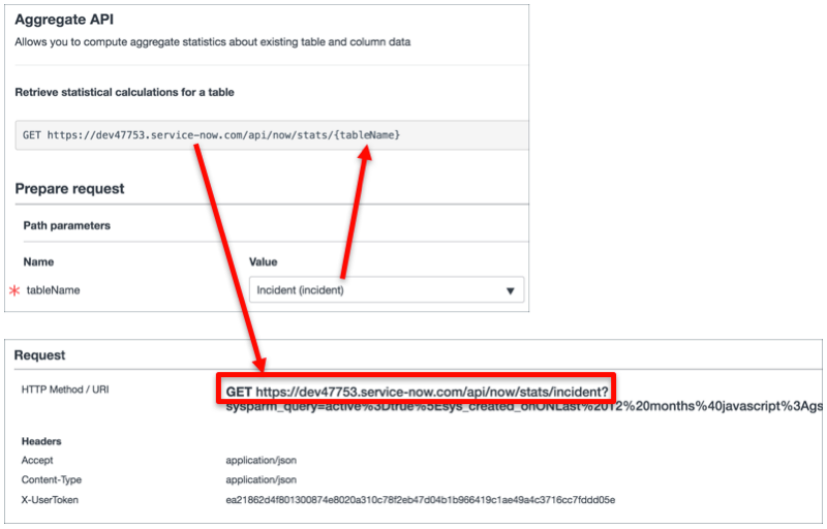


2.8 ServiceNow API Request

The REST API Explorer constructs the request to send to the ServiceNow API using the settings configured by the developer.

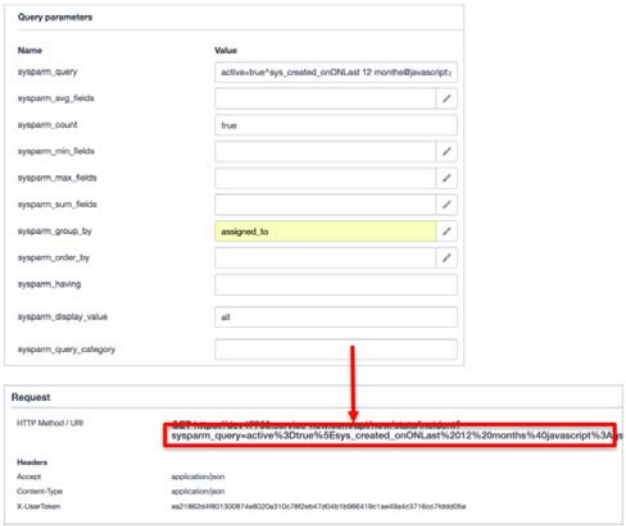
Path Parameters in the Request

The Request section displays the HTTP Method / URI to send to the ServiceNow web service. The method is from the selected API. The path parameter values are set when configuring the request.



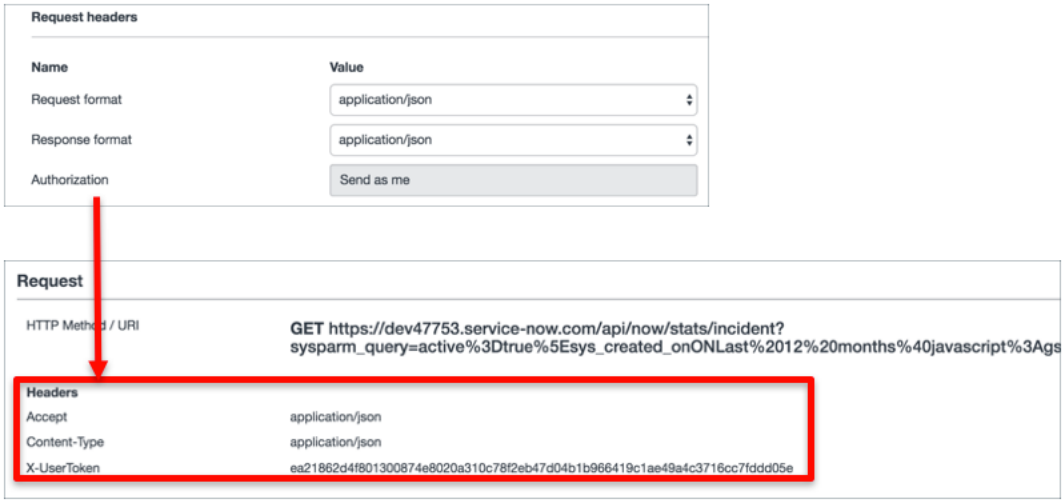
Query Parameters in the Request

The query parameters are added to the URI. In the example shown, the URI is truncated due to space limitations. To see the complete URI in the REST API Explorer, scroll horizontally.



Headers in the Request

REST headers are the meta-data associated with an API request and response. The Request Header settings appear in the request.



2.9 ServiceNow API Response

The ServiceNow API Response consists of:

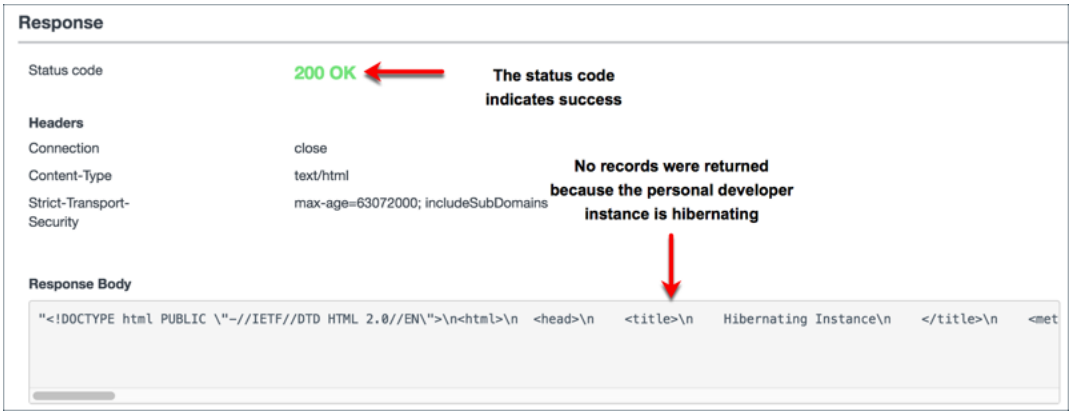
- HTTP status code
- Response headers
- Response body

HTTP Status Code

ServiceNow APIs return standard HTTP status codes. Generally speaking:

- **1xx**: Informational
- **2xx**: Success
- **3xx**: Redirection
- **4xx**: Client Error
- **5xx**: Server Error

The HTTP status codes refer to the interaction with the REST service provider. The status codes do not tell anything about the requested data. The REST transaction request can complete successfully even if no data is returned.



Response Headers

The response headers section show the returned headers and their values.

Headers	
Cache-Control	no-cache,no-store,must-revalidate,max-age=-1
Content-Encoding	gzip
Content-Type	application/json;charset=UTF-8
Date	Mon, 05 Feb 2018 23:17:45 GMT
Expires	0
Pragma	no-store,no-cache
Server	ServiceNow
Strict-Transport-Security	max-age=63072000; includeSubDomains
Transfer-Encoding	chunked
X-Is-Logged-In	true
X-Transaction-Id	03c815664f00

Response Body

The response body is the data object returned by the ServiceNow web service provider. The response body will vary depending on the selected API. In the example, the Aggregate API returns the count of open incident records in the past year with a priority of Critical or High. The results are grouped by the user in the Assigned to field.

Beth Anglin has three Critical or High priority incidents still open from the past year. Notice in this case both the value (sys_id) and display_value are included for the assigned_to field.

Data Types and Returned Values

- **Encrypted text**: The database value is encrypted, while the displayed value is unencrypted based on the user's encryption context.
- **Reference fields**: The database value is a sys_id. The display value is the human readable name.
- **Date fields**: The database value is in UTC format, while the display value is based on the user's time zone.
- **Choice fields**: The database value may be a number, but the display value will be more descriptive.
- **Currency fields**: When performing a REST request, returned currency values are converted to the local currency based on the user's Locale. When inserting data, no conversion is performed.

2.10 Exercise: Prepare and Send an API Request to the Table API

In this exercise, you will use the REST API Explorer to prepare a request for the ServiceNow Table API. You will create a request which returns all active Incident table records with the string **email** in the Short description field. You will test the API request.

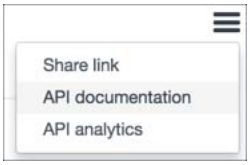
Preparation - Get the Encoded Query

1. In the main ServiceNow browser window, use the Application Navigator to open **Incident > Open**.
2. Click the **filter** icon ( ) in the Incidents list breadcrumbs to open the Condition Builder.
3. Create the conditions shown:

4. Click the **Run** button.
5. Notice how many records are returned and examine their Short descriptions.
6. Right-click on the **Short description contains email** part of the breadcrumbs and select **Copy query** from the menu.

Configure the Table API Query

1. In the main ServiceNow browser window, use the Application Navigator to open **System Web Services > REST > REST API Explorer**.
2. In the REST API Explorer, configure the API:
   Namespace: **now**
   API Name: **Table API**
   API Version: **v2**
   Method (this field does not have a label): **Retrieve records from a table (GET)**
3. Configure the Path parameters:
   tableName: **Incident (incident)**
4. Configure the Query parameters:
   sysparm_query: **paste in the encoded query you copied in the Preparation section of this Exercise**
   sysparm_display_value: **true**
   sysparm_limit: **10 (Limited to 10 results for testing)**
5. Read about the possible performance implications of setting sysparm_display_value to true.
   a. Click the **REST API Explorer** menu and select the **API documentation** menu item.



   b. On the docs site, click the appropriate ServiceNow version link.
   c. On the Table API docs site page, select **Table API - GET, /now/table/{tableName}** from the **On this page** list on the right of the page.
   d. Scroll to the **sysparm_display_value** explanation and read the **Note**. The personal developer instances have small data sets so setting this value to **true** will not cause performance issues.
6. Return to the REST API Explorer and add fields to the sysparm_fields list.

a. Click the **Edit** button ( ) on the sysparm_fields field.

b. When the slushbucket opens, start typing **Number** or scroll down until you see the **Number** field.

c. Click the **Number** field then click the **Add** button ( ).

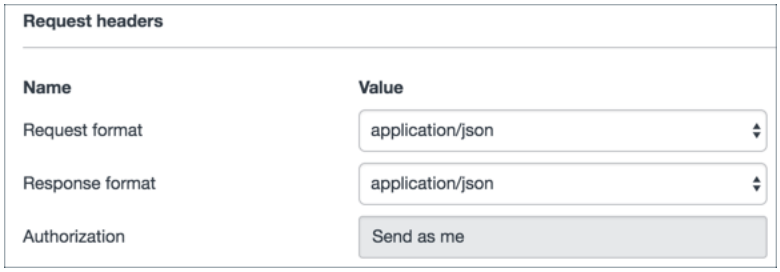
d. Also add the **Caller**, **Short description**, and **Priority** fields.

e. Click the **Save** button.

QUESTION: Why are the field names different in the sysparm_fields list different than what you selected in the slushbucket? If you aren't sure, scroll to the Answers section at the bottom of this page.

Configure the Request Headers

1. In the REST API Explorer, scroll to the **Request Headers** section and verify the header fields are set as shown:



QUESTION: In the Preparation section of this exercise, you noted how many records were returned by the query. Do you expect the same number of records to be returned by this API request? Explain your reasoning. If you aren't sure, scroll to the Answers section at the bottom of this page.

Test the API Request

1. Click the **Send** button.
2. When the response is returned, examine the Status code. It should be 200.
3. The returned result is an array of objects. Each of the objects is a record from the Incident table. Examine the response body to see which records were returned. Are these the records you expected?
4. In the returned records, the caller_id object contains a link. For one of the returned records, copy the link and paste it into a new tab in your browser. What record does the link open?

Test the sysparm_exclude_reference_link Query Parameter

1. Scroll back to the Query parameters in the REST API Explorer and set the **sysparm_exclude_reference_link** parameter to **true**.
2. Click the **Send** button again.
3. Return to the response body and examine the format of the response.

> QUESTION: How has the response body changed with sysparm_exclude_reference_link set to true? If you aren't sure, scroll to the Answers section at the bottom of this page.

Test the sysparm_fields Query Parameter

1. Scroll back to the Query parameters in the REST API Explorer and delete the contents of the **sysparm_fields** parameter.

2. Click the **Send** button again.
3. Return to the response body and examine the format of the response.

> QUESTION: Are any fields returned for the Incident table records when the sysparm_fields query parameter has no value? If so, which fields? If you aren't sure, scroll to the Answers section at the bottom of this page.

Answers

> **Question**: Why are the field names different in the sysparm_fields list different than what you selected in the slushbucket??
> **Answer**: The slushbucket displays field labels. The sysparm_fields list displays the field names.

> **Question**: In the Preparation section of this exercise, you noted how many records were returned by the query. Do you expect the same number of records to be returned by this API request?
> **Answer**: The REST API Explorer will return a maximum of ten incident records as configured. If your query returned fewer than ten records, your REST query will return all of the records from the query. If your query returned more than ten records, you will not see all of the records.

Question: How has the response body changed with sysparm_exclude_reference_link set to true?
Answer: The caller_id property of the returned record object is now a string and not an object.

Response Body (sysparm_exclude_reference_link = true)


```
{
  "result": [
    {
      "number": "INC0000047",
      "short_description": "Issue with email",
      "caller_id": "Joe Employee",
      "priority": "3 - Moderate"
    },
```


Response Body (sysparm_exclude_reference_link = false)


```
{
  "result": [
    {
      "number": "INC0000047",
      "short_description": "Issue with email",
      "caller_id": {
        "display_value": "Joe Employee",
        "link": "https://dev[redacted].service-now.com/api/now/table/sys_user/681ccaf9c0a8016400b98a06818d57c7"
      },
      "priority": "3 - Moderate"
    },
```

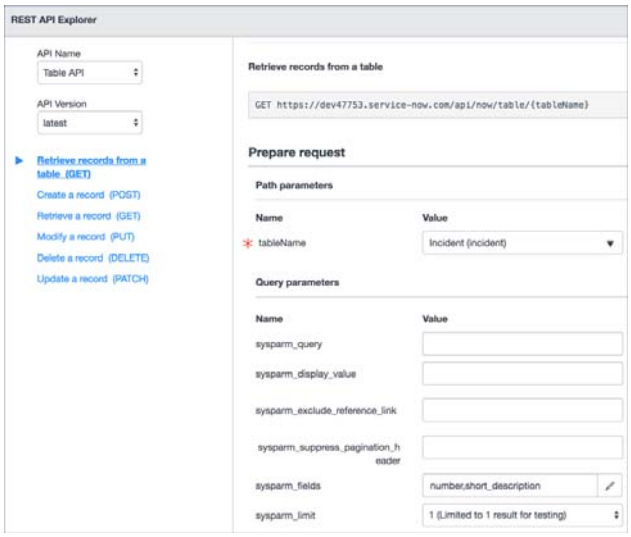

Question: Are any fields returned for the Incident table records when the sysparm_fields query parameter has no value? If so, which fields?
Answer: If the sysparm_fields query parameter has no value, all fields are returned for the record.

2.11 Exercise: Add a Header to Obtain Session Debug

In this exercise, you will add the X-WantSessionDebug header to the Incident table GET request.

Preparation - Create a Query

1. If you do not still have the REST API Explorer open from the last exercise, open it now.
2. Create this query for the Table API:



Add a Header

1. In the REST API Explorer, scroll to the Request headers section.
2. Click the **Add header** button.
3. Configure the new header:
   Header name: **X-WantSessionDebugMessages**
   Header value: **true**

4. For this header to return results, session debug must also be enabled. Enable session debugging for SQL queries.
   a. Open ServiceNow in a new tab (so you do not lose the configuration you just did in the REST API Explorer).
   b. Use the Application Navigator to enable **System Diagnostics > Session Debug > Debug SQL**. A script will be enabled.

Test the Header

1. Return to the REST API Explorer and click the **Send** button.
2. Examine the **session** object in the response body.
3. Return to the main ServiceNow browser window and disable session debugging by selecting **System Diagnostics > Session Debug > Disable All** in the Application Navigator.
4. Enable security debugging by selecting **System Diagnostics > Session Debug > Debug Security** in the Application Navigator.
5. Return to the REST API Explorer and send the request again.
6. Examine the **session** object in the response body. Access control evaluation, for example, is part of the session object.

7. Return to the main ServiceNow browser window and disable session debugging by selecting **System Diagnostics > Session Debug > Disable All** in the Application Navigator.

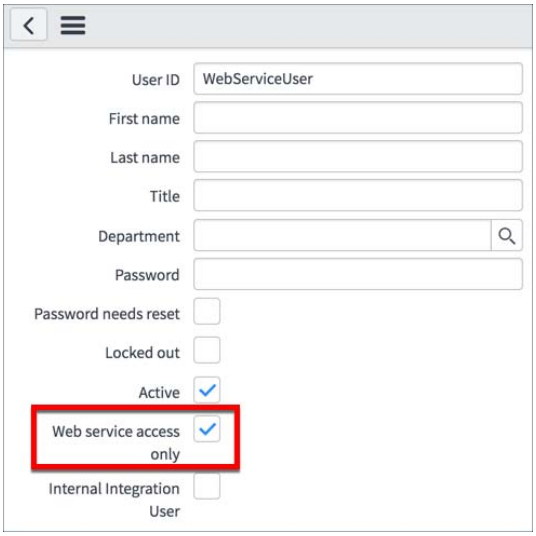
2.12 Adding Security to Inbound Requests

Strategies for adding security to inbound API requests include:

- Create a user specifically for inbound requests
- Disallow web service access to tables
- Create CORS rules

Create an API Request User

Users with the **Web service access only** option set on their user record cannot log into the ServiceNow UI. This option allows the user credentials to be used only to authorize API connections. To set this option, open the user record for editing using the **User Administration > Users** module.



The API request user must be granted the roles necessary to access the records requested by the API requests.

Disallow Web Service Access to Tables

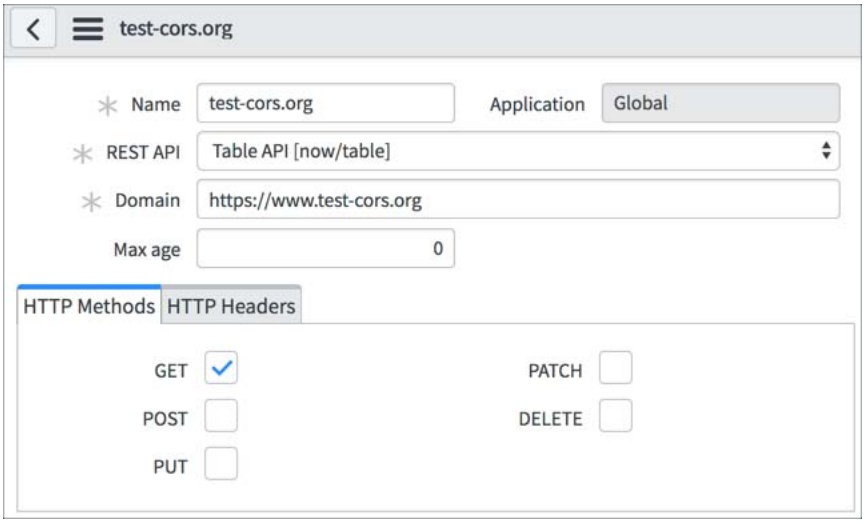
Administrators can disable web service access to tables. On the table record, open the Application Access section and de-select the **Allow access to this table via web services** option. REST requests are not accepted for tables unless this option is selected (checked). To set this option, open the table record for editing using the **System Definition > Tables** module.

> IMPORTANT: The REST API Explorer ignores this setting. The REST API Explorer can interact with tables with the Allow access to this table via web services option disabled.

2.13 CORS Rules

Cross-origin resource sharing (CORS) rules control which domains can access specific REST API endpoints. To create a CORS rule, use the Application Navigator to open **System Web Services > REST > CORS Rules**.

In the example, the resource **https://www.test-cors.org** (https://www.test-cors.org) can only access the Table API using the GET method.



- **REST API**: The REST API the CORS rule applies to.
- **Domain**: The domain for the CORS rule. Specify the domain using an IP Address or a domain pattern.
- **HTTP Methods**: Select the allowed methods.
- **HTTP Headers**: Enter a comma-separated list of HTTP headers to send in the response. Specified headers are added to the Access-Control-Expose-Headers header.
- **Max age**: Enter the number of seconds to cache the client session. After an initial CORS request, further requests from the same client within the specified time do not require a preflight message. If you do not specify a value, the default value of 0 indicates that all requests require a preflight message.

There are a number of requirements for specifying the domain including:

- Start with http:// or https://
- Must be an IP address or domain pattern
- Can contain only one wildcard *

Check out the complete list of **CORS domain requirements** (https://docs.servicenow.com/bundle/kingston-application-development/page/integrate/inbound-rest/reference/r_CORSDomainRequirements.html) on the docs.servicenow.com site.

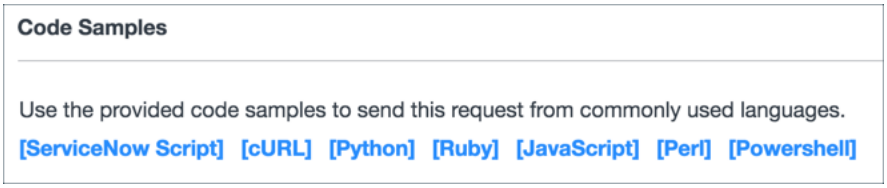
> IMPORTANT: CORS Rules cannot be tested in the REST API Explorer.

2.14 Code Samples

The REST API Explorer creates code sample for integrating to the ServiceNow APIs in several commonly used languages:

- ServiceNow Script
- cURL
- Python
- Ruby
- JavaScript
- Perl
- Powershell

To create the code sample, click the link in the REST API Explorer.



To highlight the code sample for copying, click the **Select Snippet** button.

After highlighting the code sample, copy the code sample to the clipboard.

The code samples all use fake credentials. Before using the script in the application integrating to ServiceNow, update the code to use valid credentials.

2.15 Inbound Integrations Module Recap

Core concepts:

- Use the REST API Explorer to create and test inbound ServiceNow API requests
- Path parameters are part of the endpoint URL
- Query parameters determine which records, which data, and the data format returned in the response body
- The HTTP status code indicates the status of the transaction request and does not indicate any information about the returned data
- The response body format is set in the headers
- The REST API Explorer tests requests as the currently logged in user
- The **Allow access to this table via web services** option cannot be tested in the REST API Explorer
- Do not use the admin user in code integrations; create a web services only user
- Disable web service access to tables with sensitive data unless web service access is required
- CORS rules add security to APIs
    - CORS rules determine which cross-origin resources can access which methods

- CORS rules cannot be tested in the REST API Explorer

- Code samples provide script stubs for integrating into ServiceNow from third party applications
  - Developers must update credentials in the code samples
  - Code samples are available in multiple standard languages used with integrations

3. Scripted REST APIs

In this module you will create a Scripted REST API. You will test the API using the REST API Explorer.

3.1 Scripted REST API Objectives

In this module, you will learn to:

- Create a Scripted REST API
  - Define query parameters
  - Create API documentation
  - Add request headers
  - Set response and request types
  - Define resources (GET, POST, PUT, PATCH, DELETE)

- Create a Scripted REST API Resource
  - Specify a path including path parameters
  - Add query parameter associations
  - Write a Resource Script using the request and response objects
  - Return a standard error object
  - Create and return custom error objects

- Enable Scripted REST API versioning
  - Set a default version
  - Create new API versions
  - Test versions in the REST API Explorer

- View API metrics on the Usage by WEB API dashboard

IMPORTANT: The module assumes familiarity with **Inbound Integrations** (https://developer.servicenow.com/app.do#!/training/article/app_store_learnv2_rest_kingston_inbound_integrations/app_store_learnv2_rest_kingston_inbound v=kingston), including the REST API Explorer.

3.2 Exercise: Prepare Instance for Scripted REST Services

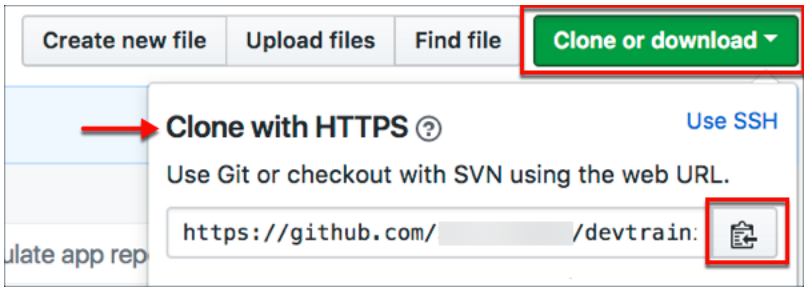
In this module, you will create a scripted REST service that allows third-party applications to interact with the NeedIt application data. The NeedIt application, which is used to request services and goods from several departments, is already partially built for you. You will load the application into your Personal Developer Instance from a source code repository on GitHub.com.

> **NOTE**: For more details about the process detailed in this exercise or if you have two-factor authentication enabled on your GitHub account, visit the **GitHub and ServiceNow section** (https://developer.servicenow.com/app.do#!/program/faq?q=github_and_servicenow) of the Developer Program FAQ.

> **IMPORTANT**: If you have already forked the NeedIt repository and created the NeedIt application from source control for other training modules, skip to the **Create a Branch from a Tag** section.

Fork the NeedIt Repository

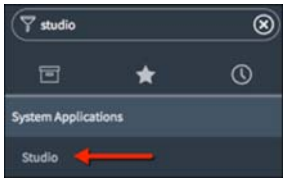
1. In a web browser, navigate to **the NeedIt Repository** (https://github.com/ServiceNow/devtraining-needit-kingston.git) .
2. If you have a GitHub account, log in. If you do not have an account, create an account and log in. When creating an account, you may have to navigate to the GitHub repository in step 1 again.
3. Click the **Fork** button (Fork) to create a copy of the repository in your GitHub account.
4. GitHub automatically loads the forked repository page. Copy the URL for the forked repository.
   a. Click the **Clone or download** button.
   b. Examine the URL. If the URL contains the word ServiceNow, this is not your forked copy of the repository. The URL should contain your GitHub user name.
   c. Make sure **Clone with HTTPS** is selected. If not, click the **Use HTTPS** link.



   d. Click the **Copy to clipboard** button ( ).

Import an Application from the Forked Repository

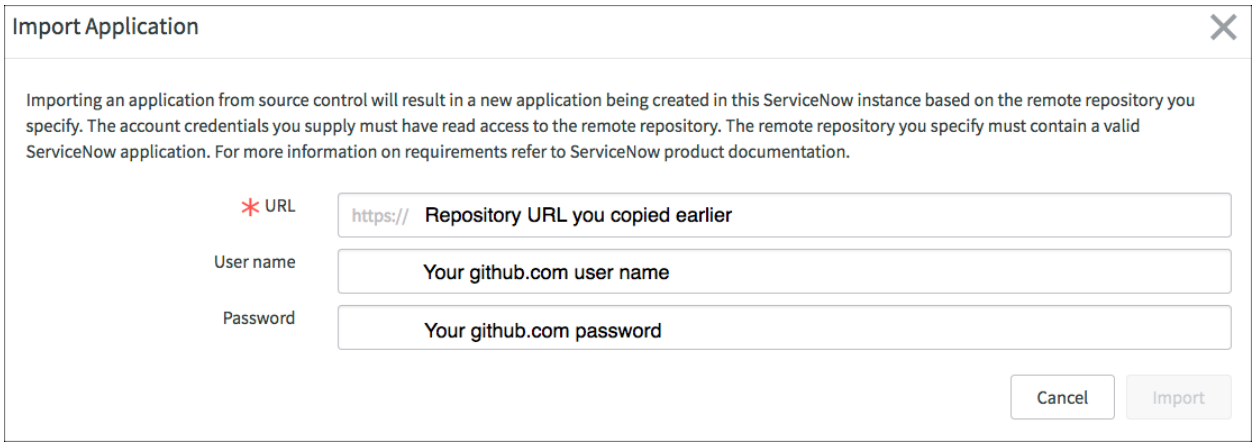
1. Log in to your ServiceNow Personal Developer instance (PDI) as the admin user. If you do not have a PDI, log in to **the ServiceNow Developer Portal** (https://developer.servicenow.com) and request a **kingston** instance.
2. In the Application Navigator, open **System Applications > Studio**.



3. Click the **Import from Source Control** button.

4. Configure the connection to the **Forked** source control repository.

   URL: **<URL you copied for your forked version of the repository>**
   User name: **<Your github.com user name>**
   Password: **<Your github.com password>**



> **IMPORTANT**: Make sure you connect Studio to YOUR FORKED REPOSITORY. The URL syntax should be something like: **https://github.com/YourGitHubUser/devtraining-needit-kingston** (https://github.com/YourGitHubUser/devtraining-needit-kingston). DO NOT connect to the original ServiceNow repository. The ServiceNow repository is read only. Your forked repository is read and write.

5. Click the **Import** button. A progress dialog appears.

6. When the application import is completed, click the **Select Application** button.

7. In the Load Application dialog, click the link to the **NeedIt** application to open it for editing in Studio. You will not see any application files in Studio until you create a branch from a tag in the next section of this exercise.

Create a Branch from a Tag

1. In Studio, open the Source Control menu and click the **Create Branch** option.
2. Configure the Branch.

   Branch Name: **ImportData**
   Create from Tag: **LoadForScriptedWebServicesModule**

> **NOTE**: If you do not have a **LoadForScriptedWebServicesModule** tag in your repository, it means you forked the repository before this tag was added. Create a branch from the **Solution_ExtendGlideAjax** tag instead.

3. Click the **Create Branch** button.

4. Click the **Close Dialog** button.
5. To load UI changes from the branch, return to the main ServiceNow browser window (not Studio) and use the browser's reload button to refresh the page.

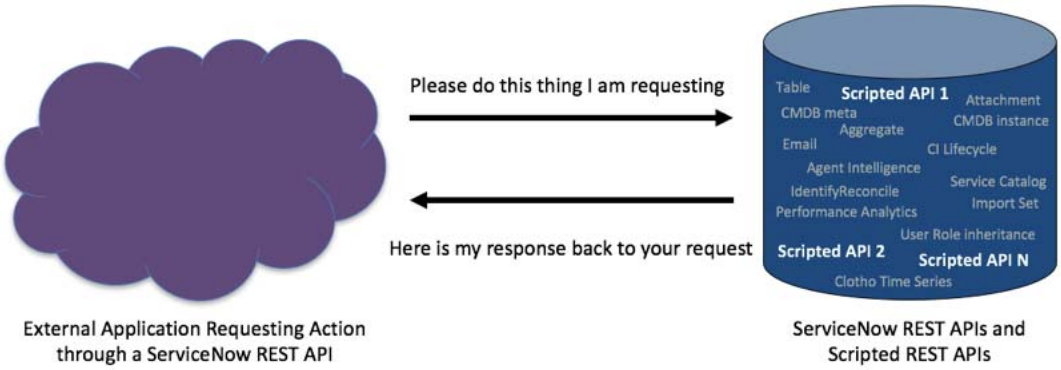
> **IMPORTANT**: If the branch creation fails and you see a message about confirming the user has read, write, and create branch access, it means you connected Studio to the ServiceNow version of the repository and not your forked version of the repository. Another possible cause is that you have two-factor authentication enabled on your GitHub account. Visit the **GitHub and ServiceNow section** (https://developer.servicenow.com/app.do#!/program/faq?q=github_and_servicenow) of the Developer Program FAQ for details on how to troubleshoot this error or configure Studio with two-factor authentication enabled.

3.3 What Are Scripted Web Services?

Web services make it possible for applications to connect to other software applications over a network allowing an exchange of information between the provider (server) and client (consumer).

A web service consumer (client) requests information from a web service provider (server). A web service provider processes the request and returns a status code and a response body. When the response body is returned, the web service consumer extracts information from the response body and takes action on the extracted data.

ServiceNow provides APIs (https://docs.servicenow.com/bundle/kingston-application-development/page/build/applications/concept/api-rest.html) which make it easy for developers to request information from ServiceNow in third-party applications or from other ServiceNow instances. In many cases, the ServiceNow APIs provide the methods developers need for their integrations. In other cases, custom APIs are required. Scripted REST Services allow developers to create their own APIs on the Now Platform.



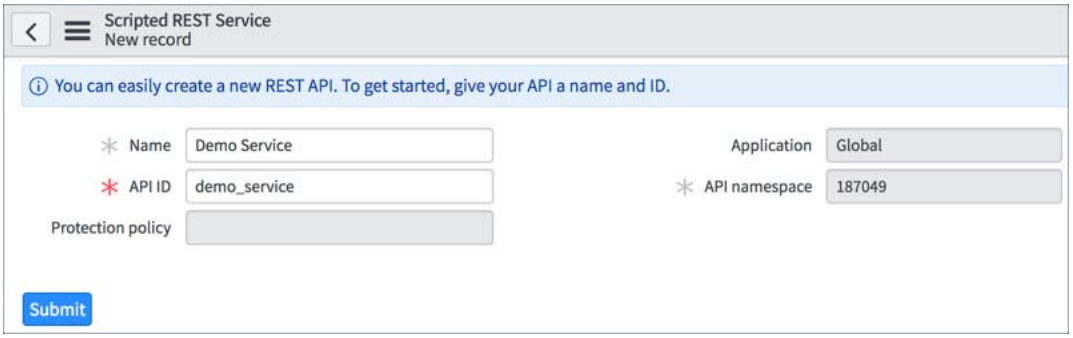
3.4 Creating Scripted REST APIs

The procedure for adding files to an application in Studio is the same regardless of file type:

1. Click the **Create Application File** button.
2. Choose the new file type, in this case, **Scripted REST API**.
3. Configure the new file.

To create a Scripted REST API using the main ServiceNow browser window, use the Application Navigator to open **System Web Services > Scripted Web Services > Scripted REST APIs**. Click the **New** button.

Configure the Scripted REST Service record.



- **Name**: Web service name.
- **API ID**: Used as part of the API path when the API is invoked. ServiceNow constructs a default value for this field based on the value in the Name field. Developers can change the default value. Do not use spaces and avoid special characters other than underscore (_).

Click the **Submit** button to continue configuring the scripted REST service.

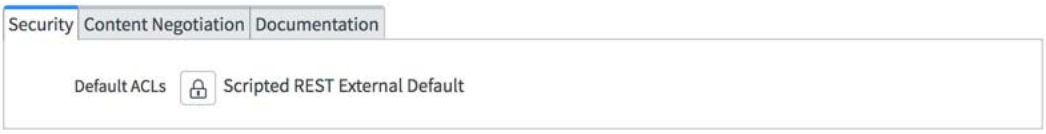
Scripted REST APIs do not take actions; they are containers which contain resources. Resources are the actions third-party applications invoke.

3.5 Security, Content Negotiation, and Documentation

The Security, Content Negotiation, and Documentation sections are available after saving a Scripted REST API for the first time.

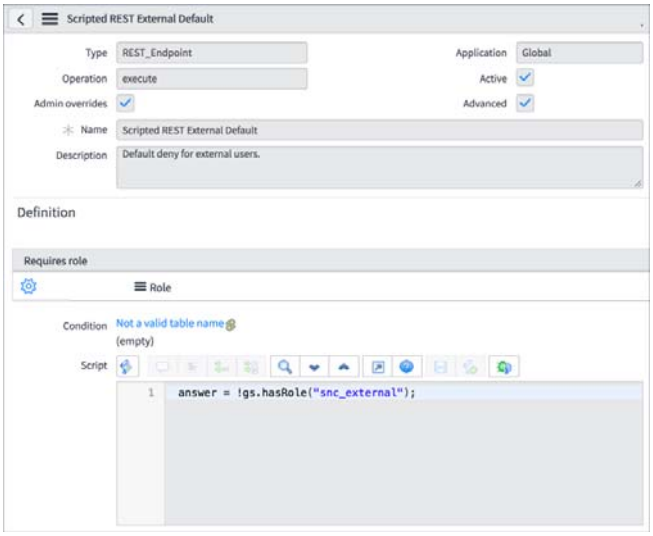
Security

A Scripted REST API may require users to pass an Access Control check. A requesting user must satisfy at least one of the Access Controls. It is not necessary to satisfy all selected Access Controls. A default Access Control, Scripted REST External Default, is applied to all new Script REST APIs. Developers can remove the default and/or add Access Controls of their choice. Click the **Unlock** button to modify the Default ACLs field value.



Access Controls for Scripted REST APIs, have the Access Control type **REST_Endpoint**.

The default Access Control denies access to the Scripted REST API to any user with the snc_external role.



Content Negotiation

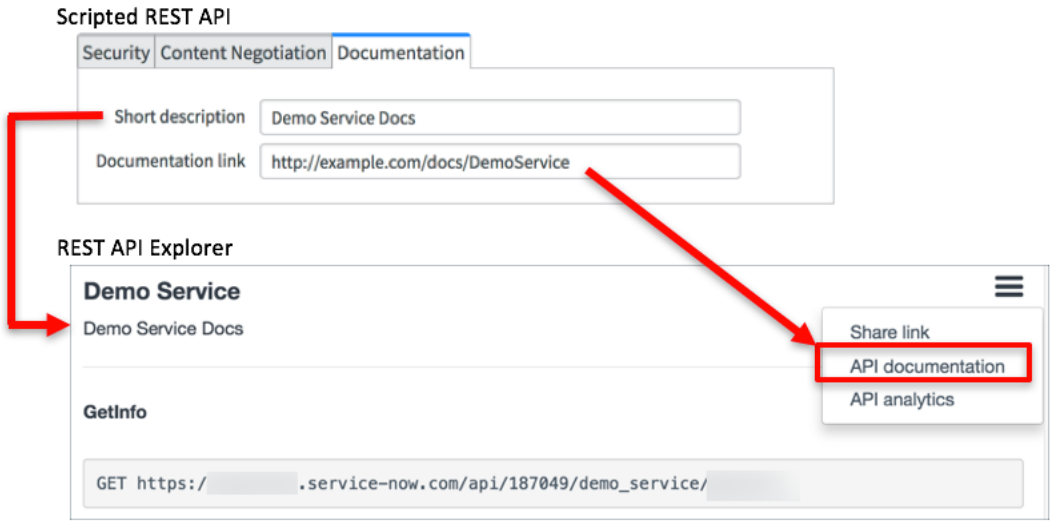
The Content Negotiation section defines the supported formats for the request and response formats. The default for both the request and response is to allow:

- application/json
- application/xml
- text/xml

To change the list of supported formats, click the Override default option for the request or response format.

Documentation

Use the Documentation section to direct users to documentation about the Scripted REST API. When the API is selected in the REST API Explorer, the Short description field value is displayed at the top of the form. Users access the documentation link from the REST API Explorer menu.



DEVELOPER TIP: Provide a meaningful description of the API's function in the Short description field to help other developers understand the purpose of the API.

3.6 Request Headers and Query Parameters

Request Header

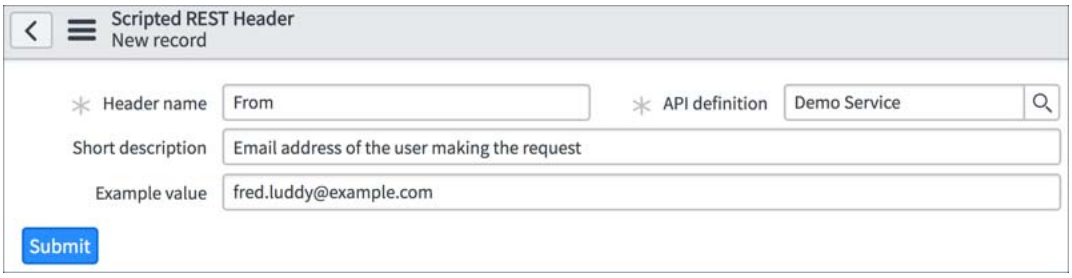
A REST request header contains parameters (metadata) which define the HTTP(S) interaction. Commonly used REST headers include:

- Authorization
- Accept

- Content-Type

See **List of HTTP Header Fields** (https://en.wikipedia.org/wiki/List_of_HTTP_header_fields) for a list of HTTP header fields.

In the Request Headers related list, click the **New** button to add a request header to the API.

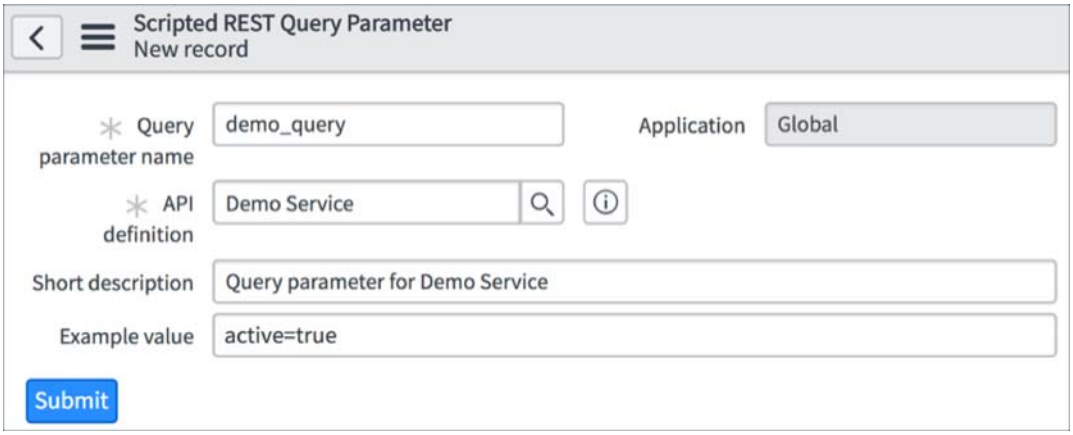


- **Header name**: HTTP header field name.
- **API definition**: API the header is part of.
- **Short description**: Describe what information should be passed in the header.
- **Example value**: Demonstrate how to use the header by providing a sample of the data to be passed.

Request headers can also be defined in resources.

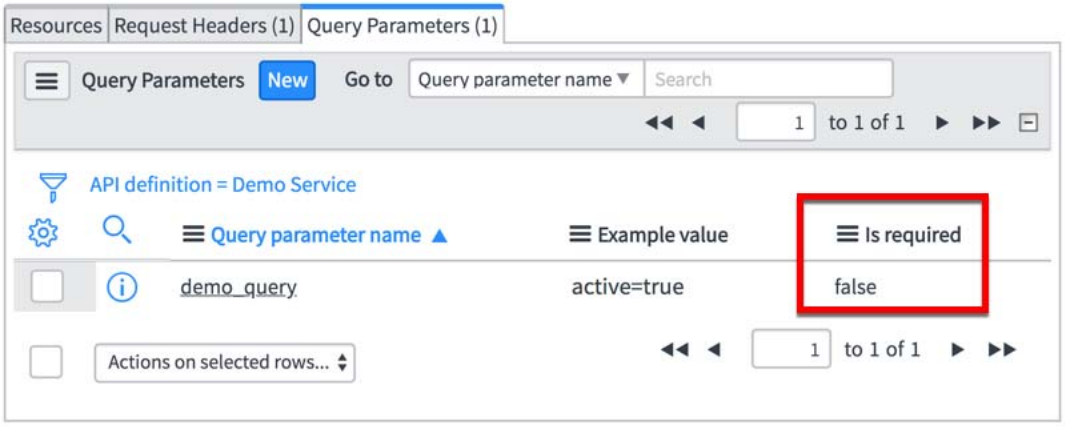
Query Parameters

Query parameters control what information developers using the API can pass in the API request URL. In the Query Parameters related list, click the **New** button to create a parameter.



- **Query parameter name**: Name of the parameter. By convention, query parameters are lowercase and use underscores in place of spaces. Many query parameters in baseline ServiceNow Scripted REST APIs start with the string sysparm_.
- **API definition**: API the query is part of.
- **Short description**: Describe what information should be passed in the query.
- **Example value**: Demonstrate how to use the query by providing a sample of the data to be passed.

After configuring a query parameter, set the Is required value in the Query Parameters related list. Set the value to true if the query parameter is mandatory. The default value is false.



Query parameters can also be defined in resources.

3.7 Exercise: Create the NeedIt App REST Service

In this exercise, you will create a scripted web service for third-party applications to interact with the NeedIt application data.

Create a Scripted REST API - NeedIt API

1. Open the NeedIt application for editing in Studio.
    a. In the main ServiceNow browser window, use the Application Navigator to open **System Applications > Studio**.
    b. In the Load Application dialog, click the **NeedIt** application.
2. Create a Scripted REST API.
    a. In Studio, click the **Create Application File** button.
    b. In the Filter... field enter the text **REST** OR select **Inbound Integrations** from the categories in the left hand pane.
    c. Select **Scripted REST API** in the middle pane as the file type then click the **Create** button.
3. Configure the Scripted REST API:
    Name: **NeedIt API**
    API ID: **needit_api**
4. Click the **Submit** button.

Security and Content Negotiation

1. Scroll to the **Security** section (tab). ServiceNow added a default ACL to the record. Leave the default.
2. Switch to the **Content Negotiation** section (tab) and modify the default configuration.
    Override default supported request formats: **Selected (checked)**
    Default supported request formats: **application/json**
3. Click the **Update** button.

Documentation

1. Create a UI Page to document the NeedIt API.
    a. In Studio, click the **Create Application File** button.
    b. In the Filter... field enter the text **UI** OR select **Forms & UI** from the categories in the left hand pane.

c. Select **UI Page** in the middle pane as the file type then click the **Create** button.

2. Configure the UI Page.
   Name: **needit_api_documentation**
   Category: **General**
   Description: **Documentation for the NeedIt App API**

3. Copy this HTML to your clipboard:

```
<h1 style="color: #5e9ca0;"><span style="color: #000080;">NeedIt Scripted API</span></h1>
<p>The NeedIt Scripted API provides methods for interacting with data from the NeedIt application including: </p>
<ul>
<li>GET ni_getinfo: Get aggregate (count) information for records for a specific user from the NeedIt table.  The records are grouped by Request type.  The response includes the total number of NeedIt records for the user (integer), the count of NeedIt records for each Request type (array of objects), and user information (object).</li>
</ul>
```

4. In the HTML field, paste the copied HTML between the jelly tags.

5. Click the **Submit** button.

6. Right-click on the Endpoint field value and select the menu item to copy a URL from the menu (the menu item name is browser dependent).

7. In Studio, return to the **NeedIt API** tab. If you closed the record, use the Application Explorer to open **Inbound Integrations > Scripted REST APIs > NeedIt API**.

8. Configure the **Documentation** section (tab).
   Short description: **Provides access to the NeedIt application record data.**
   Documentation link: **Paste in the Endpoint you copied in a prior step**

9. Click the **Update** button.

Query Parameters

1. Switch to the **Query Parameters** related list and click the **New** button.

2. Configure the query:
   Query parameter name: **ni_query**
   Short description: **Encoded query for searching for specific NeedIt records**
   Example value: **active=true**

3. Click the **Submit** button.

3.8 Scripted REST API Resources

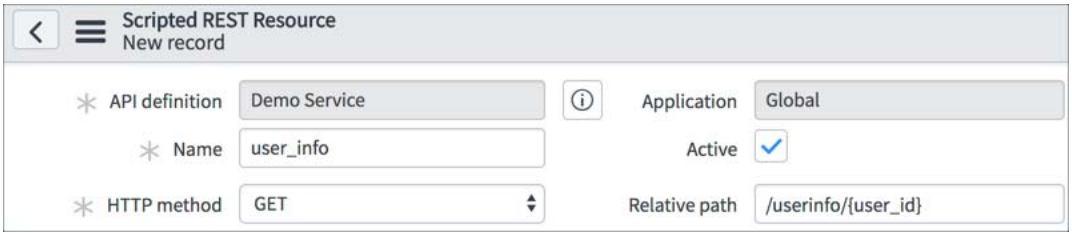
A REST API is a collection of REST resources. REST resources are unique data representations which have at least one URI. Resources are typically a set of related information such as a record, changes to a record, or calculations based on records. Each resources is defined by an HTTP method:

- GET
- POST
- PUT
- PATCH

- DELETE

Creating Resources

In the Scripted REST API record, open the **Resources** related list. Click the **New** button then configure the resource.



- **Name**: Name of the resource. By convention, resource names are lowercase with underscores (_) instead of spaces.
- **HTTP method**: Select GET, PUT, POST, PATCH, or DELETE.
- **Relative path**: Specify the unique part of the URI. The Relative path is appended to the Base API path defined in the Scripted REST API. Enclose path parameters in { }. Users pass values for path parameters in the service's URL.

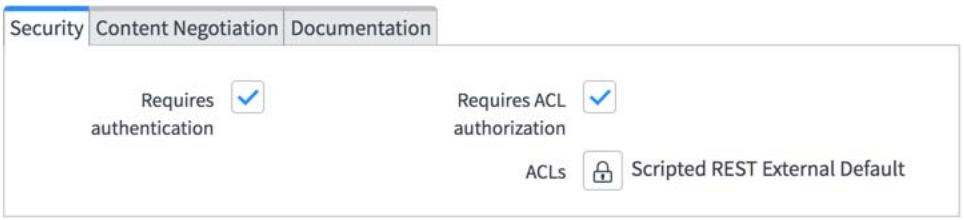
The Resource path field displays after the new record is saved for the first time. The Resource path is the Base API path concatenated with the Relative path.

3.9 Resource Security, Content, and Documentation

Resources can override settings inherited from the Scripted REST API.

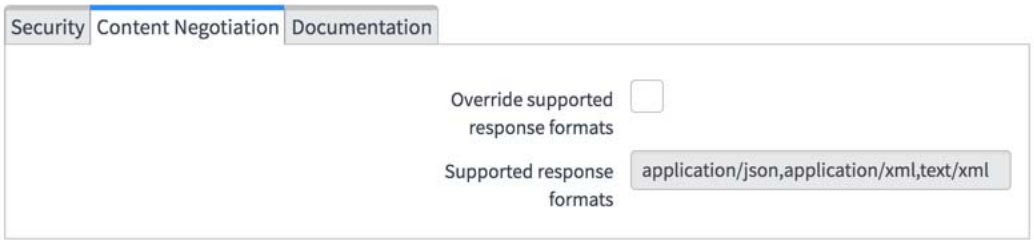
Security

By default, resources require authentication. To disable authentication, de-select (uncheck) the **Requires authentication** option. To disable the Access Control check, de-select (uncheck) the **Requires ACL authorization** option.



Content Negotiation

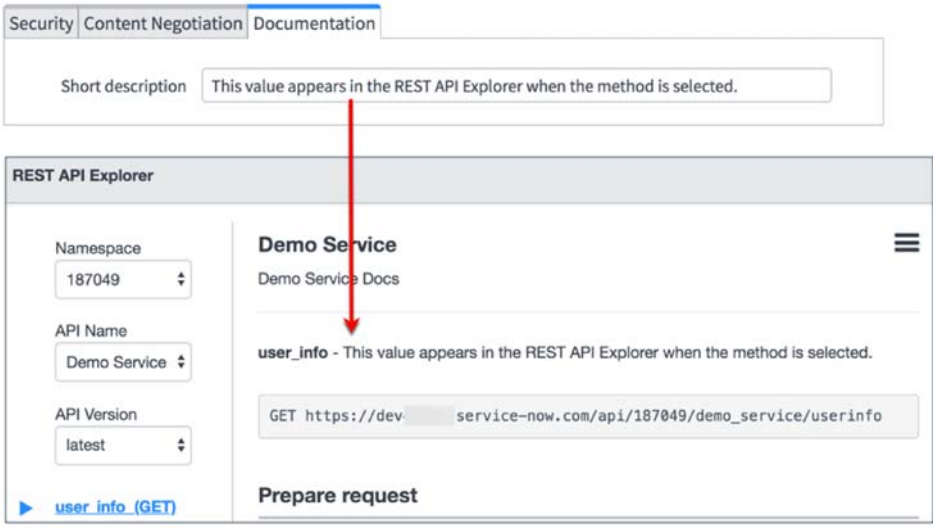
Resource request format is inherited from the Scripted REST API and cannot be overwritten. To change allowed response formats, select the Override supported response formats then list the formats in the Supported response formats field.



DEVELOPER TIP: If multiple Access Controls are listed, requesting users must satisfy only one Access Control and not all.

Documentation

The Short description field value appears in the REST API Explorer when the resource is selected. Use this field to provide useful information to developers using the resource.

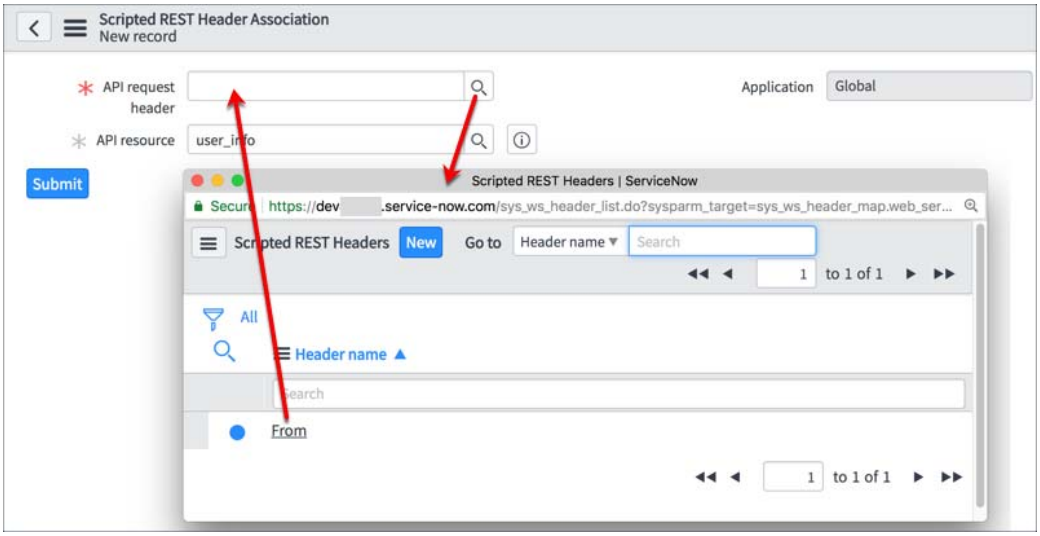


3.10 Resource Request Header and Query Parameter Associations

Request headers and query parameters are defined in the Scripted REST API and associated with resources in the resource definitions.

Request Header Associations

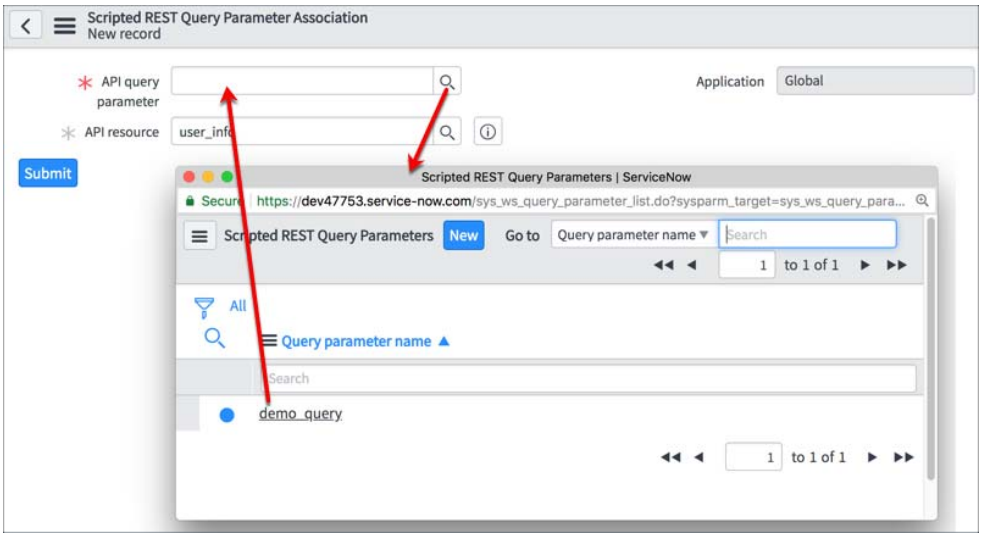
To associate a request header, scroll to the Request Headers Associations related list. Click the **New** button. Use the API request header field Search button to select a request header to associate with the resource.



Click the **New** button in the Scripted REST Headers dialog to create a new request header.

Query Parameter Associations

To associate a query parameter, scroll to the **Query Parameters Associations** related list. Click the **New** button. Use the API query parameter field Search button to select a query parameter to associate with the resource.



Click the **New** button in the Scripted REST Query Parameters dialog to create a new query parameter.

3.11 Exercise: Create the ni_getinfo Resource

In this exercise, you will create a GET resource in the NeedIt API Scripted Web Service. The resource will be passed a user and an encoded query.

Create a GET Resource

1. If not still open from the last exercise, open the NeedIt application for editing in Studio.
    a. In the main ServiceNow browser window, use the Application Navigator to open **System Applications > Studio**.
    b. In the Load Application dialog, click the **NeedIt** application.
2. If not still open from the last exercise, use the Application Explorer to open **Inbound Integrations > Scripted REST APIs > NeedIt API**.

3. Switch to the **Resources** related list and click the **New** button.
4. Configure the Resource:
   Name: **ni_getinfo**
   HTTP method: **GET**
   Relative path: **/nigetinfo/{user_name}**
5. Click the **Submit** button.

Query Parameter Associations

1. Scroll to the **Query Parameter Associations** related list and click the **New** button.
2. Configure the API query parameter association:
   API query parameter: **ni_query**
3. Click the **Submit** button.
4. Return to the **ni_getinfo** Scripted REST Resource tab in Studio.
5. Click the **Additional Actions** menu ( ) and select the **Reload form** menu item.
6. Scroll to the **Query Parameter Associations** related list. The ni_query query parameter should be associated to the Resource.
7. Make the ni_query parameter required.
   a. Double click in the **Is required** column for the ni_query query parameter in the Query Parameter Associations related list.
   b. Set the Is required value to **true**.
   c. Click the **Save** button ( ).
8. Click the **Update** button.

Documentation

1. Switch to the **Documentation** section (tab).
2. Configure the **Documentation** section (tab).
   Short description: **Find the NeedIt active NeedIt records for a user. Group by What needed.**
3. Click the **Update** button.

3.12 Resource Script

A resource script is server-side JavaScript that creates and populates properties on the response object. The response object is returned to the application which invoked the resource.

The resource script field contains a script template. The template's process function, known as a self-invoking or immediately invoked function expression, is automatically invoked when a resource is accessed. The request and response objects, which are passed to the process function, are automatically created.

Developers add server-side JavaScript to the template after the comment.

3.13 Resource Script - RESTAPIRequest

The RESTAPIRequest API allows developers to access data from the request. The *request* object is automatically instantiated from the RESTAPIRequest class and passed to the process function in the Scripted REST API script.

The properties of the request object are:

- body
- pathParams
- queryParams
- queryString
- uri
- url
- headers
- getHeader()
- getSupportedResonseContentTypes()

Body

The **body** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest_body) object provides access to the request body.

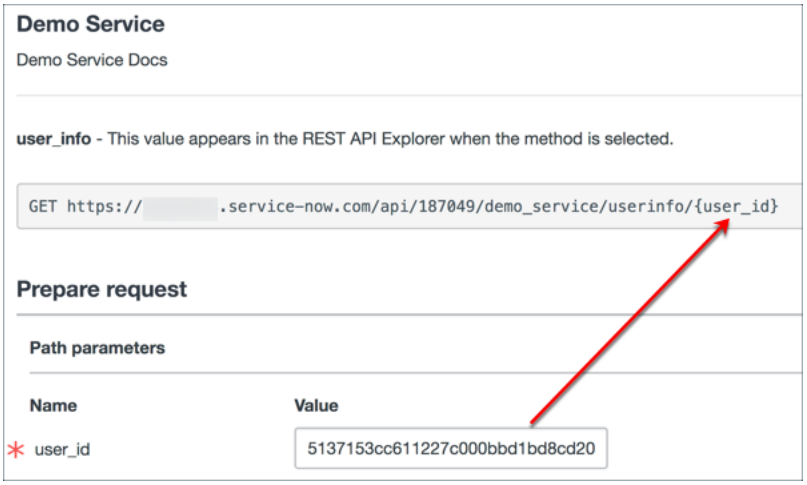
```
//get instance of RESTAPIRequestBody
var requestBody = request.body;
```

pathParams

The **pathParams** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest_pathParams) object allows script access to path parameters passed in the request URL. The available path parameters are determined by the Scripted REST Service resources. The userinfo resource URL from the Demo Service is:

```
https://<instance>/api/187049/demo_service/userinfo/{user_id}
```

The path parameters are passed in when the service is invoked.



```
//get pathParams object
var pathparams = request.pathParams;
//get user_id property value from pathparams object
var userID = pathparams.user_id;
```

The value of the pathparams.user_id property after the example script executes is the same as the value passed in the URL.

```
Information    pathparams.userID= 5137153cc611227c000bbd1bd8cd2005d
```

queryParams

The **queryParams** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest_queryParams) object allows script access to the query parameters from the web service request. The Demo Service query parameter is demo_query. The Demo Service URL is:

```
https://<instance_rest_endpoint>/?demo_query=active%3Dtrue
```

The demo_query parameter value is passed in when the service is invoked.


```
//get queryParams object
var queryparams = request.queryParams;
//value of myQueryParam is active=true
var myQueryParam = queryparams.demo_query;
```

The value of the myQueryParam variable after the example script executes is **active=true**.

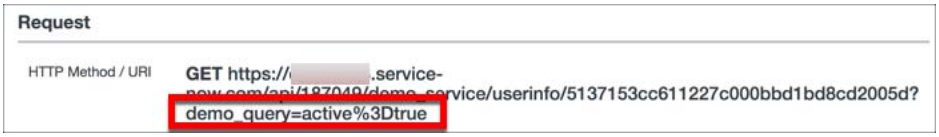
```
Information    demo_query = active=true
```

queryString

The **queryString** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest_queryString) string contains the entire query added to the endpoint URI. The Demo Service URL is:

```
https://<instance>/api/187049/demo_service/userinfo/5137153cc611227c000bbd1bd8cd2005d?demo_query=active%3Dtrue
```

The query string is part of the URL.




```
//get the query string

//value of query is demo_query=active%3Dtrue
var query = request.queryString;
```

The value of the query variable after the example script executes is **demo_query=active%3Dtrue**.

```
Information   query = demo_query=active%3Dtrue
```

uri

The **uri** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest_uri) string contains the request URI, excluding domain information. The userinfo resource URL from the Demo Service is:

```
https://<instance>/api/187049/demo_service/userinfo/{user_id}
```

The URI does not include the query parameters.


```
//get the uri string
//value of query is /api/187049/demo_service/userinfo/5137153cc611227c000bbd1bd8cd2005d
var query = request.uri;
```

The value of the query variable after the example script executes is **/api/187049/demo_service/userinfo/5137153cc611227c000bbd1bd8cd2005d**.

url

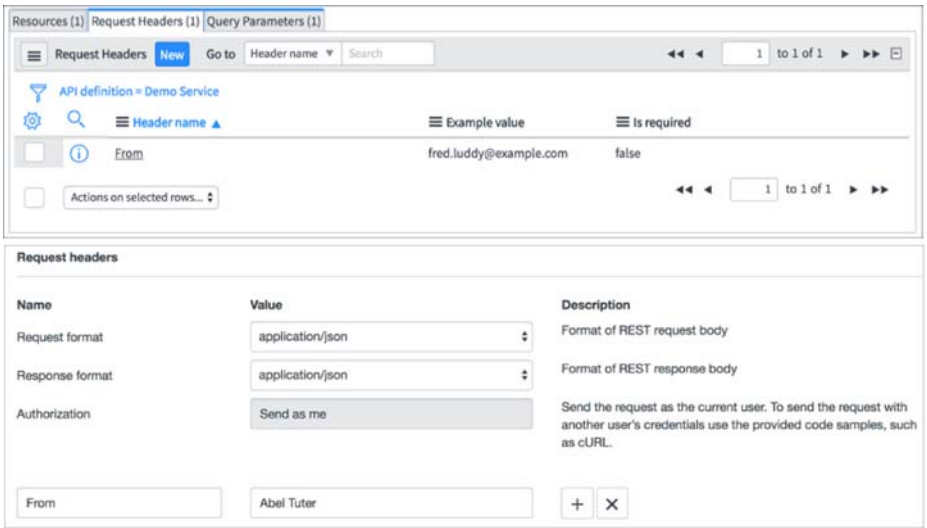
The **url** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest_url) string contains the entire request URL.

```
//get the url string
//returns https://instance/api/187049/demo_service/userinfo/5137153cc611227c000bbd1bd8cd2005d
var query = request.url;
```

The value of the query variable after the script executes is the complete URL.

headers

The **headers** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest_headers) object contains all headers property value pairs from the request.

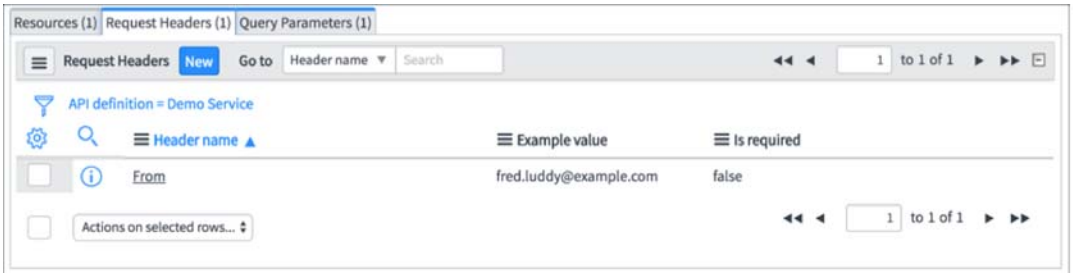


```
//get the headers from the request
var headers = request.headers;
//get the value of the Accept property
var acceptHeader = headers.accept;
```

The properties of the request.headers object can be different for different APIs. The request.headers properties for the Demo Service API include accept, from, and content-type. Notice that all request.headers property names are lowercase even if defined in the API using uppercase characters. The script used to log the request.headers property also logged the data type. Notice that all properties are strings.

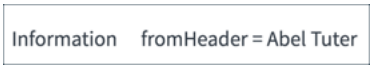
getHeader()

The **getHeader** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_RESTAPIRequest-geviceRequestHeader_String) method returns a string containing the value of a specific header from the web service request.




```
//get the headers from the request
var headers = request.headers;
//get the value of the from header
var fromHeader = headers.getHeader('from');
```

The variable fromHeader has the value Abel Tuter after the script executes.



DEVELOPER TIP: Header property names are lowercase even if the property is defined in the API using uppercase characters.

getSupportedResponseContentTypes()

The **getSupportedResponseContentTypes()** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=r_SSR-getSupportedRespContentTypes) method returns an array of string values where each string is a content type, such as application/json.

```
var contentTypes = [];
    contentTypes = request.getSupportedResponseContentTypes();
    for(i=0;i<contentTypes.length;i++){

        gs.info("content type ["  + i + "] = " + contentTypes[i]);
    }
```

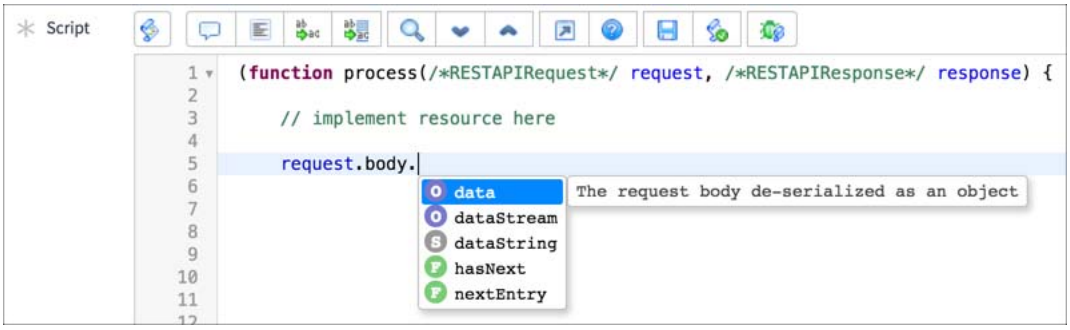
The contentTypes array has one element for the Demo Service.

```
Information    content type [0] = application/json
```

3.14 Resource Script - RESTAPIRequestBody

The RESTAPIRequestBody API allows developers to access the body content of a scripted REST API request. The request.body object is instantiated automatically.

To see the properties of the request.body object, in the resource Script field, type **request.body** then press **<control> + <spacebar>** on your keyboard. Hover the cursor over a property to see a description of the property.



Icons in the properties list identify the data type of the property:

- **O**: object
- **S**: string
- **F**: function

The RESTAPIRequestBody API includes:

- **data**: The request body content as a single object or array of objects.
- **dataStream**: The content of the request body as a stream.
- **dataString**: The content of the request body as a string.
- **hasNext()**: Returns true if the request body contains another entry.
- **nextEntry()**: Retrieves one entry from the request body as a script object.

View sample request bodies in the **RESTAPIRequestBody API documentation** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_RESTAPIRequestBody).

3.15 Resource Script - RESTAPIResponse

The RESTAPIResponse API allows developers build a response to a scripted REST API request. The response object is instantiated automatically.

The RESTAPIResponse API includes:

- **getStreamWriter()**: The ResponseStreamWriter for the response. Use this object to write directly to the response stream.
- **setBody()**: Create the response body, as a JavaScript object. The body content is automatically serialized to JSON or XML depending on the value of the Accept header passed in the request.

- **setContentType()**: Assigns a value, such as application/json, to the Content-Type header in the web service response.
- **setError()**: Set the properties of the response error object when an error is returned.
- **setHeader()**: Assign a value to a REST service response header.
- **setHeaders()**: Sets the headers for the web service response.
- **setLocation()**: Assigns a value to the Location header in the web service response. See the **W3 Location header documentation** (https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.30) for more information about this header.
- **setStatus()**: The status code to send in the response, such as 200 to indicate success.

The **RESTAPIResponse API documentation** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_RESTAPIResponse) includes sample scripts for the API's methods.

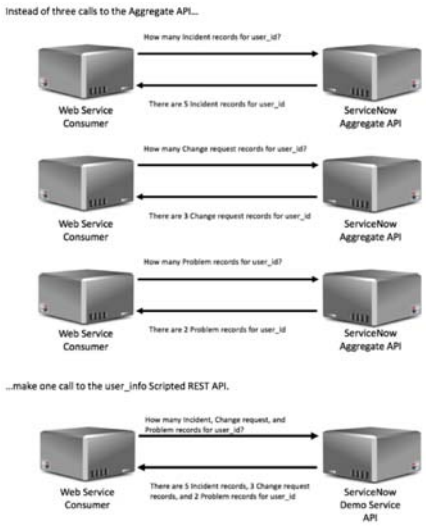
> **DEVELOPER TIP**: To write directly to the scripted REST API response stream instead of using the RESTAPIResponse API, use the **RESTAPIResponseStream API** (https://developer.servicenow.com/app.do#!/api_doc?v=kingston&id=c_RESTAPIResponseStream).

3.16 Example Resource Script

ServiceNow provides an Aggregate API that is used to compute aggregate statistics about existing table and column data. In the use case demonstrated in this module, three aggregations are required:

- Incident table records where the user_id is the Caller.
- Change request table records where the user_id is the Requested by.
- Problem table records where the user_id is the Opened by.

Instead of making three calls to the Aggregate API, a Scripted REST API performs the aggregations and returns the aggregations in a single response object.



```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

    // implement resource here


    // Get value from the user_id path parameter passed in the URL
    var requestUser = request.pathParams.user_id;
    // Get value of the demo_query query parameter passed in the URL
    var requestDemoQuery = request.queryParams.demo_query;

    // Query the sys_user table to get the user record for the user passed in
    // the user_id path parameter
    var requestUserName = new GlideRecord('sys_user');
    requestUserName.get(requestUser);

    // Aggregation 1: Incident table
    // Get the count of Incident table records where the user from the user_id path
    // parameter is the Caller.
    var userIncidentCount = new GlideAggregate('incident');
    userIncidentCount.addAggregate('COUNT');
    userIncidentCount.addQuery('caller_id',requestUser);
    userIncidentCount.addEncodedQuery(requestDemoQuery);
    userIncidentCount.query();

    var incidents = 0;
    if (userIncidentCount.next()) {
        incidents = userIncidentCount.getAggregate('COUNT');
    }


    // Aggregation 2: Change request table
    // Get the count of Change request table records where the user from the user_id path
    // parameter is the Requested by.
    var userChangeCount = new GlideAggregate('change_request');
    userChangeCount.addAggregate('COUNT');
    userChangeCount.addQuery('requested_by',requestUser);
    userChangeCount.addEncodedQuery(requestDemoQuery);
    userChangeCount.query();

    var changes = 0;
    if (userChangeCount.next()) {
        changes = userChangeCount.getAggregate('COUNT');
    }


    // Aggregation 3: Problem table
    // Get the count of Problem table records where the user from the user_id path
    // parameter is the Opened by.
    var userProblemCount = new GlideAggregate('problem');
    userProblemCount.addAggregate('COUNT');
    userProblemCount.addQuery('opened_by',requestUser);
    userProblemCount.addEncodedQuery(requestDemoQuery);
    userProblemCount.query();

    var problems = 0;
    if (userProblemCount.next()) {
        problems = userProblemCount.getAggregate('COUNT');
    }


    //Create a body object.  Add property value pairs to the body.
    var body = {};
        body.numInc = incidents;
        body.numChg = changes;
        body.numPrb = problems;
        body.user = {"User name": requestUserName.user_name,
            "User ID": requestUser};

    // Send the response object, which is returned to the requestor, to the body object.
    response.setBody(body);

    })(request, response);
```

The response body contains the number of incidents, changes, and problems, as well as the user's name and ID.

Response Body


```
{
  "result": {
    "numInc": "5",
    "numChg": "2",
    "numPrb": "0",
    "user": {
      "User name": "fred.luddy",
      "User ID": "5137153cc611227c000bbd1bd8cd2005"
    }
  }
}
```


Some Things to Remember about Resource Scripts

- The process function is self-invoking.
- The request and response objects are automatically instantiated.
- Access controls apply to Scripted REST APIs. The user making the request through the API's authentication must have access to requested information.

3.17 Exercise: Add Script to the ni_getinfo Resource

In this exercise, you will add a script to the ni_getinfo resource. The script will find the number of active NeedIt requests for the user passed in the path parameter. The response object consists of:

- Count of active NeedIt records for user (number)
- Count of active NeedIt records for the user by Request type (array of objects)
- Name and sys_id of user (object)

You will test the ni_getinfo resource using the REST API Explorer.

Preparation

1. In the main ServiceNow browser window (not Studio), use the Application Navigator to open **NeedIt > Open**.
2. If not already on the list, add the **Requested for** and **Request type** columns.
   a. Click the **Personalize list** icon ().
   b. In the Available slushbucket, double-click the **Requested for** field to move it to the Selected slushbucket.
   c. Use the **Move up** ( ) and **Move down** ( ) buttons to place the Requested for field at the location of your choice.
   d. Add the **Request type** field to the Selected slushbucket at the location of your choice.
   e. Click the **OK** button.
3. Choose a user such as Beth Anglin or Fred Luddy. For each Request type (Human Resources, Facilities, and Legal), verify there is at least one record in which the user of your choice is the Requested for. If necessary, modify existing records or create new records.
4. In the NeedIt record list, use the Condition Builder to find the number of active records for the user of your choice. The example searches for Fred Luddy. Be sure to search for the user of your choice.

5. Make a note of how many active NeedIt records have your user as the Requested for.

Resource Script

1. If the ni_getinfo resource is not still open for editing, in Studio use the Application Explorer to open **Inbound REST Integrations > Scripted REST Resources > NeedIt API/ni_getInfo[GET]**.
2. Replace the contents of the Script field with this script:

```
(function process(/*RESTAPIRequest*/ request, /*RESTAPIResponse*/ response) {

// implement resource here


// Get value from the user_name path parameter passed in the URL
var requestUser = request.pathParams.user_name;
// Get value of the ni_query query parameter passed in the URL
var requestNIQuery = request.queryParams.ni_query;

// Query the sys_user table to get the user record for the user passed in
// the user_id path parameter.
var requestUserName = new GlideRecord('sys_user');
requestUserName.get('user_name',requestUser);



// Get the count of active NeedIt table records where the user from the
// user_name path parameter is the Requested for.  The encoded query is from the
// Query Parameter Associations.  Group by category.
var userNICount = new GlideAggregate('x_58872_needit_needit');
userNICount.addAggregate('COUNT');
userNICount.addQuery('u_requested_for',requestUserName.sys_id);
userNICount.addEncodedQuery(requestNIQuery);
userNICount.groupBy('u_request_type');
userNICount.query();


var needitRecs = 0;
var niUserRecs = [];
var needitRecsCount = 0;

while (userNICount.next()) {
    needitRecs = userNICount.getAggregate('COUNT');
    var reqType = userNICount.u_request_type.getDisplayValue();
    niUserRecs.push({"requestType": reqType, "count": needitRecs});
    needitRecsCount = parseInt(needitRecsCount) + parseInt(needitRecs);
}

//Create a body object.  Add property value pairs to the body.
var body = {};
body.totalUserNIRecs = needitRecsCount;
body.catCounts = niUserRecs;
body.user = {"User name": requestUserName.user_name,
"User ID": requestUserName.sys_id};

// Send the response object, which is returned to the requestor, to the body object.
response.setBody(body);

})(request, response);
```

3. Read through the script to make sure you understand what it does.
4. Click the **Update** button.

Test the ni_getinfo Resource

1. In the main ServiceNow browser window (not Studio), use the Application Navigator to open **System Web Services > REST > REST API Explorer**.

2. Configure the REST API Explorer to test the NeedIt API's ni_getinfo resource.

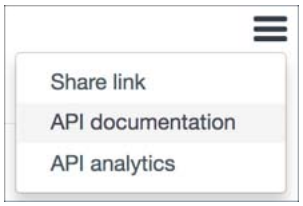
   Namespace: **x_58872_needit**
   API Name: **NeedIt API**
   API Version: **latest**

3. Examine the documentation in the REST API Explorer.

   > **QUESTION**: Where was the text **Provides access to the NeedIt application record data** defined? If you are not sure, scroll to the Answers section at the bottom of this page.

   > **QUESTION**: Where was the text **ni_getinfo - find the active NeedIt records for a user. Group by What needed** defined? If you are not sure, scroll to the Answers section at the bottom of this page.

4. Open the **API menu** and select the **API documentation** menu item.



   > **QUESTION**: Where was the API documentation defined? If you are not sure, scroll to the Answers section at the bottom of this page.

5. Prepare the request:

   user_name: **<user name of the user from the Preparation section. For example, fred.luddy>**
   ni_query: **active=true**

6. Scroll to the **Request headers** section.

   a. Examine the choice list for the **Request format**.

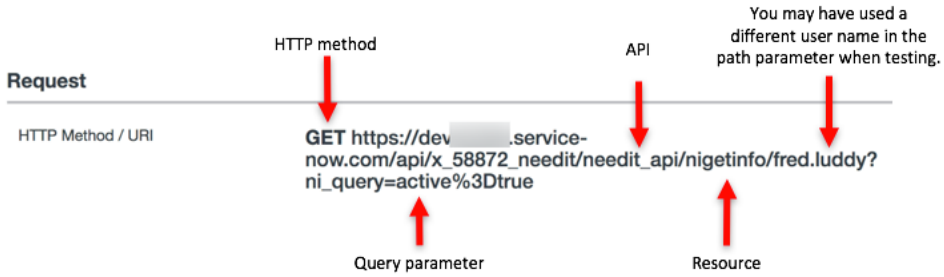
   b. Examine the choice list for the **Response format**.

   > **QUESTION**: Why is application/json the only choice in the Request format field? If you are not sure, scroll to the Answers section at the bottom of this page.

7. Click the **Send** button.

Examine the Test in the REST API Explorer

1. In the REST API Explorer, scroll to the Request section. Notice the format of the request. Can you locate the path parameter and the query parameter?



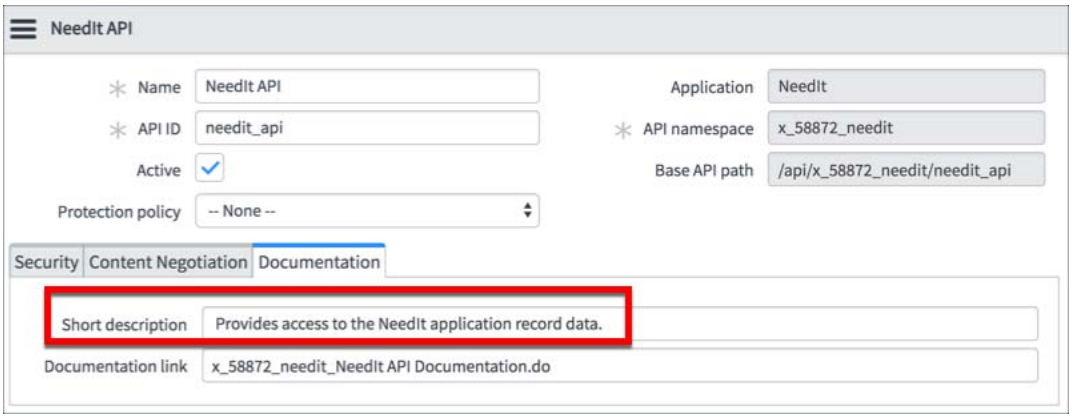
2. Scroll to the Response section and examine the Status code and Headers. The Status code should be 200 OK.

3. Examine the Response Body. It should look something like this (your user and counts may be different than the example):

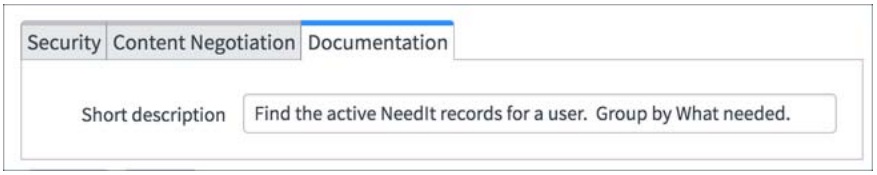

```
{
  "result": {
    "totalUserNIRecs": 6,
    "catCounts": [
      {
        "requestType": "Facilities",
        "count": "1"
      },
      {
        "requestType": "Human Resources",
        "count": "2"
      },
      {
        "requestType": "Legal",
        "count": "3"
      }
    ],
    "user": {
      "User name": "fred.luddy",
      "User ID": "5137153cc611227c000bbd1bd8cd2005"
    }
  }
}
```


Answers

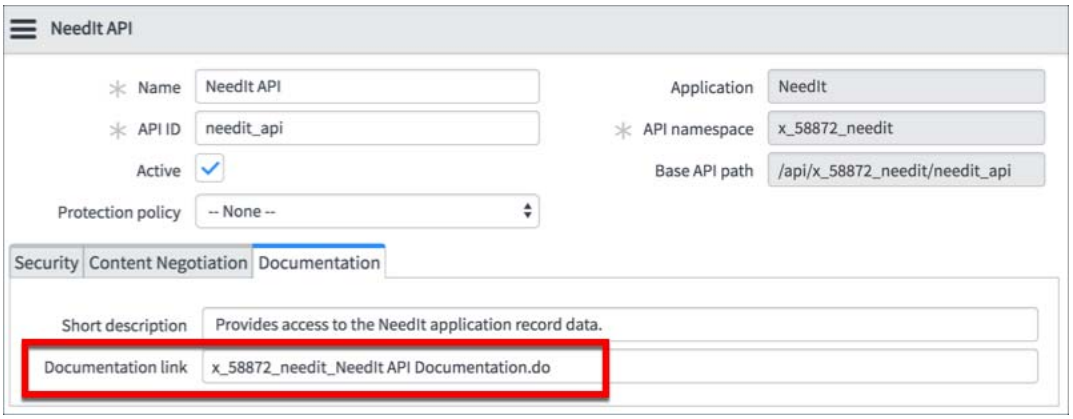
Question: Where was the text **Provides access to the NeedIt application record data** defined?
Answer: The text was defined in the Content section (tab), Short description field in the NeedIt API.



Question: Where was the text **ni_getinfo - find the active NeedIt records for a user. Group by What needed** defined?
Answer: The REST API Explorer inserted the method name. The text was defined in the Documentation section (tab), Short description field in the NeedIt API ni_getinfo Resource.



Question: Where was the API documentation defined?
Answer: The API documentation was defined as a UI Page and referenced in the Content section (tab), Documentation link field in the NeedIt API.



Question: Why is application/json the only choice in the Request format field?
Answer: The acceptable Request formats are set in the Content Negotiation section of the NeedIt API.

3.18 Scripted REST API Error Objects

To assist API users in debugging problems, use an error object to report information about the error to an API's consumers. ServiceNow has two strategies for errors:

- Use a predefined error object
- Create a custom error object

In both cases, the error object **must be instantiated from the sn_ws_err namespace**.

Predefined Error Objects

Predefined error objects send information to the consumer using standard HTTP status codes. Pass an error message to send a custom message.

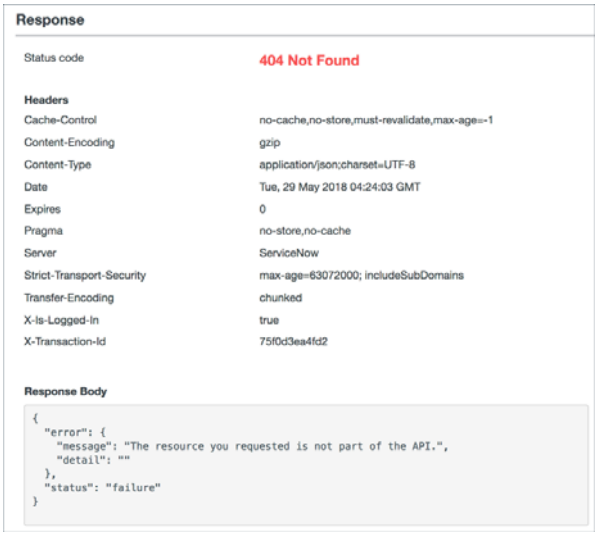
Status Code	Error Object	Description
400	BadRequestError	Error in the request, such as incorrect syntax
404	NotFoundError	Requested resource is not available
406	NotAcceptable Error	The Accept header value is not allowed
409	ConflictError	There is a conflict in the request
415	UnsupportedMediaTypeError	The request media type is not supported

For example, send a NotFoundError object to the consumer:

```
(function run(request, response) {
  // resource is not available
  return new sn_ws_err.NotFoundError('The resource you requested is not part of the API.');
})(request,response);
```

> **DEVELOPER TIP**: It is not required to send a message with the error object, but users of your API will appreciate it if you do!

The predefined error is returned in the response body.



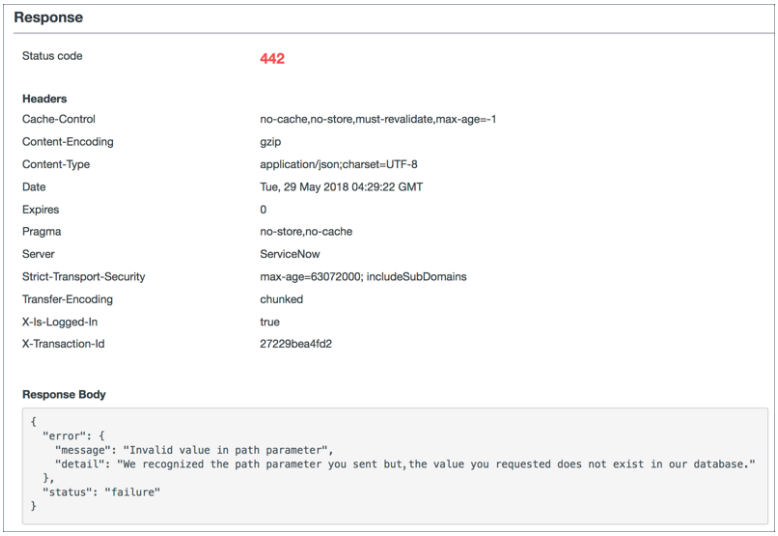
Custom Error Objects

Use the ServiceError object to define custom errors. The ServiceError methods are:

Method	Description	Default Value
setStatus(number)	HTTP Status Code for the error.	500
setMessage(string)	Short error message to send	empty string
setDetail(string)	Detailed error message	empty string

```
(function run(request, response) {
  var myError = new sn_ws_err.ServiceError();
  myError.setStatus(442);
  myError.setMessage('Invalid value in path parameter');
  myError.setDetail('We recognized the path parameter you sent, but the value you requested does not exist in our database.');
  return myError;
})(request,response);
```

The custom error is returned in the response body.



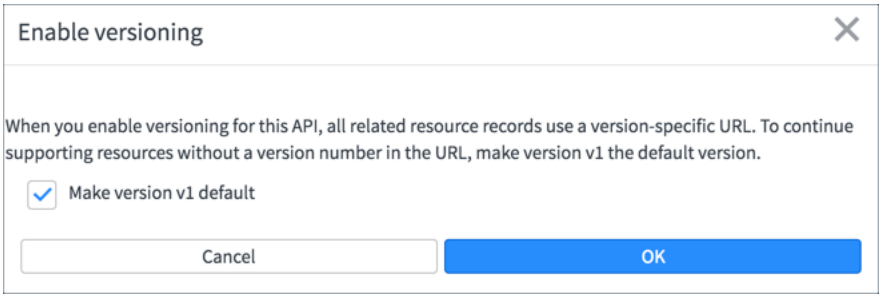
3.19 API Versions

In the default case, API versioning is disabled. Enabling versioning allows developers to test and deploy changes without impacting existing integrations from web service consumers.

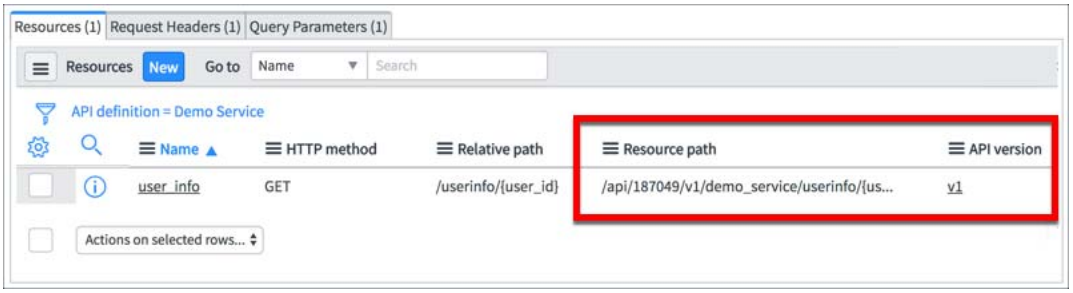
Enabling Versioning

Versioning is enabled at the API level and is applied to all API Resources. To enable versioning, click the **Enable Versioning** related link in the Scripted REST API.

When versioning is enabled, Resource URLs contain a version number. To prevent breaking existing integrations when enabling versioning, make version v1 the default.



The Resources list contains an API version column and the version number is added to the Resource path.

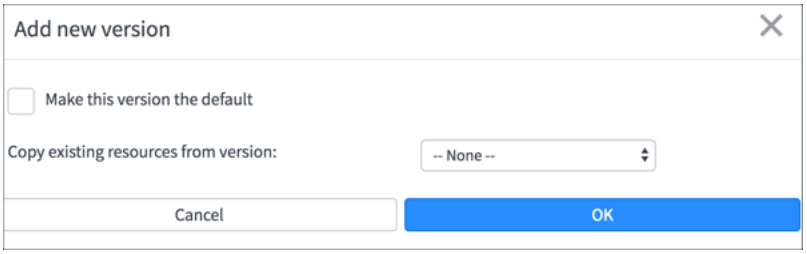


Once versioning is enabled, it cannot be disabled.

Creating a New Version

To add a new version to an API, click the **Add new version** related link.

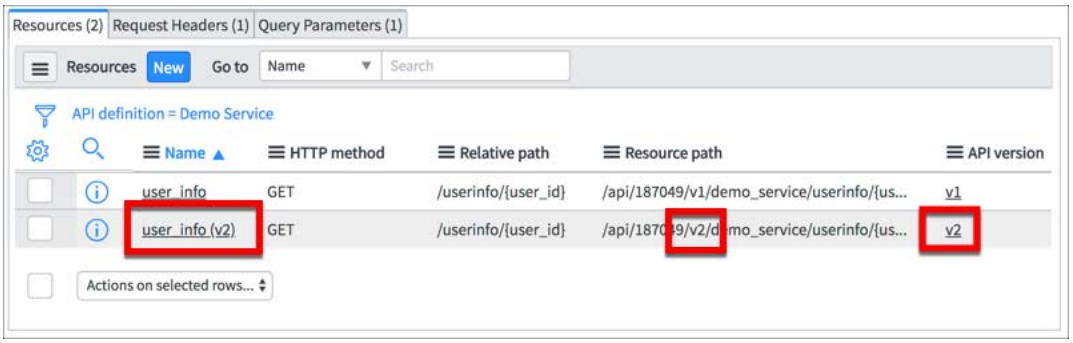
The **Add new version** dialog opens.



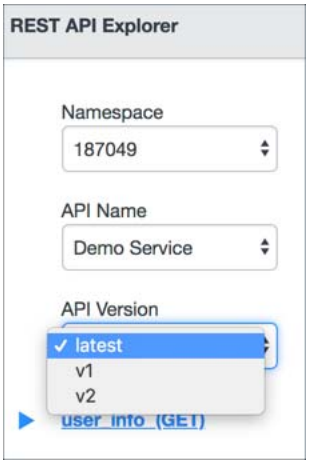
To make the new version the default, click the **Make this version the default** check box. In most cases, do not make a new version the default until development is complete and the new version is completely tested.

To copy resources from an existing version, select a version in the **Copy existing resources from version** choice list.

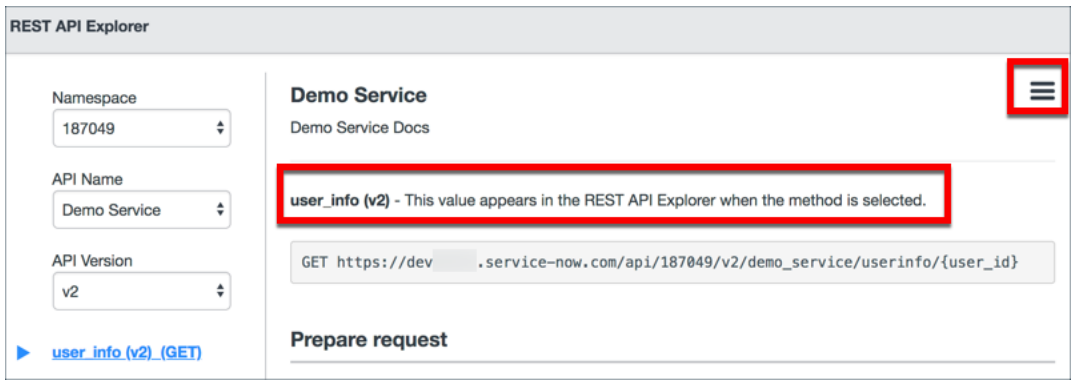
When editing resources, select the version you intend to edit.



When testing in the REST API Explorer, set the API Version to the version you want to test.



DEVELOPER TIP: Document any changes between versions in your Scripted REST APIs. Update both the method definition and the full documentation access from the REST API Explorer menu.



3.20 Exercise: Enable Versioning

In this exercise, you will enable versioning for the NeedIt API. You will create a new version of the ni_getinfo resource which returns an error object if the user passed in the user_name path parameter is not found in the sys_users table.

Enable Versioning

1. If the NeedIt API is not still open for editing, in Studio use the Application Explorer to open **Inbound Integrations > Scripted REST APIs > NeedIt API**.
2. Enable versioning.
    a. Click the **Enable Versioning** related link.
    b. When prompted in the Enable versioning dialog, select the **Make version v1 default** check box.
    c. Click the **OK** button.

Add a New Version

1. Click the **Add new version** related link.
2. Configure the new version:
    Make this version the default: **Not selected (not checked)**
    Copy existing resources from version: **v1**
3. Click the **OK** button.

Edit the New Version

1. Scroll to the Resources related list and click the link for **ni_getinfo(v2)**.
2. Examine the Resource path field and note the version number in the Resource path.
3. Modify the Script to return a custom error object if the user passed in the user_name path parameter in the request cannot be found in the sys_user table.
    a. Copy the custom error logic.

```
// If there is no user record for the user_name passed in the user_name path parameter,
// return a custom error.  Notice the error object is created from
// sn_ws_err.ServiceError();  If no status value is set, the HTTP Status will
// be 500.
if(!requestUserName.user_name){
    var apiError = new sn_ws_err.ServiceError();
    apiError.setStatus(542);
    apiError.setMessage("User not found");
    apiError.setDetail("No user record found for the user_name passed into the ni_info web service resource using the
user_name path parameter.");
    response.setError(apiError);
    return;
}
```

b. Paste the custom error logic into the script on line 16 (your line numbers may be slightly different). The error object logic goes between getting the user record from the database and creating the userNICount object.

```
11      // Query the sys_user table to get the user record for the user passed in
12      // the user_id path parameter.
13      var requestUserName = new GlideRecord('sys_user');
14      requestUserName.get('user_name',requestUser);
15
16
17
18      // Get the count of active NeedIt table records where the user from the
19      // user_name path parameter is the Requested for.  The encoded query is from the
20      // Query Parameter Associations.  Group by category.
21      var userNICount = new GlideAggregate('x_58872_needit_needit');
22      userNICount.addAggregate('COUNT');
```

c. Read through the error object logic to make sure you understand what it does.

4. Click the **Update** button.

Update the API Documentation

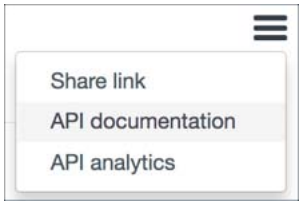
1. In Studio, use the Application Explorer to open **Forms & UI > UI Pages > NeedItAPIDocumentation**.
2. Add a list item to the HTML:

```
<li>GET ni_getinfo(v2): Has all the same features as v1 but also includes a custom error message if the request passes an invalid
user name.</li>
```

3. Click the **Update** button.

Test the ni_getinfo(v2) Resource

1. In the Resource record in Studio, click the **Explore REST API** Related Link.
2. Notice the REST API Explorer opened with the Namespace, API Name, API Version, and Resource selected.
3. Open the **API menu** and select the **API documentation** menu item. Is the new v2 documentation part of the API documentation?



4. Prepare the request:

user_name: **<Pass an invalid user name. For example, hello.world>**

ni_query: **active=true**

5. Click the **Send** button.
6. Scroll to the Response section and examine the Status code. The Status code should be 542.
7. Examine the Response Body. It should contain the custom error object.

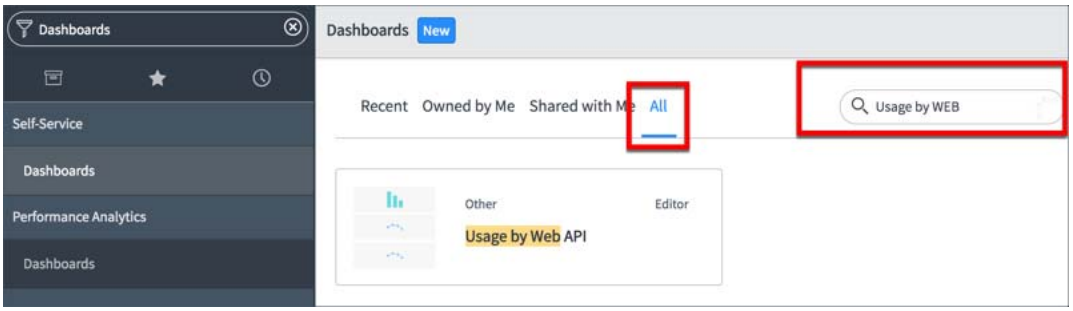
```
{
  "error": {
    "message": "User not found",
    "detail": "No user record found for the user_name passed into the ni_info web service resource using the user_name path parameter."
  },
  "status": "failure"
}
```

3.21 API Analytics

The **Usage by WEB API** dashboard displays analytic data on a per API basis. Developers can view:

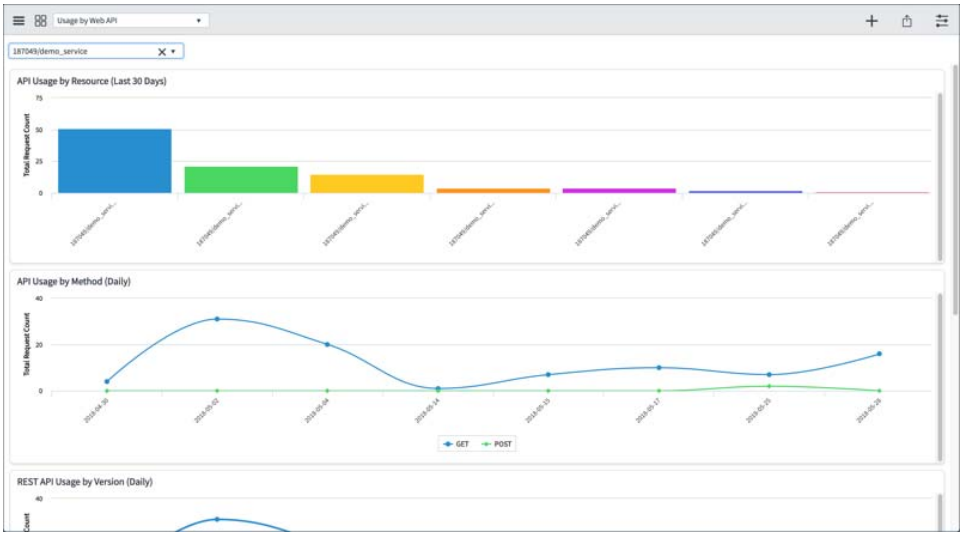
- API Usage by Resource (Last 30 Days)
- API Usage by Method (Daily)
- REST API Usage by Version (Daily)
- API Usage (Monthly)

To view the Dashboard for an API, use the Application Explorer to open **Performance Analytics > Dashboards**. Select **All**, then use the search field to look for **Usage by WEB API**. Open the Dashboard then select the desired API from the choice list.



OR, click the **API analytics** Related Link in the API record.

Use the dashboard to monitor requests to the API.



3.22 Exercise: API Dashboard

In this exercise, you will open and examine the Dashboard for the NeedIt API.

Open Dashboard

1. If the NeedIt API is not still open for editing, in Studio use the Application Explorer to open **Inbound Integrations > Scripted REST APIs > NeedIt API**.
2. Click the **API analytics** Related Link.

Explore the Dashboard

The NeedIt API and its resources have not had a lot of traffic so your dashboard will not have a lot of data. See if you can determine:

- How many times each resource has been invoked. (Hint: Click on data in the Dashboard widgets for more information.)
- Which resource was the most busy:
    - Last 30 days
    - Today
    - Monthly
- Which day was the busiest day for the NeedIt API?

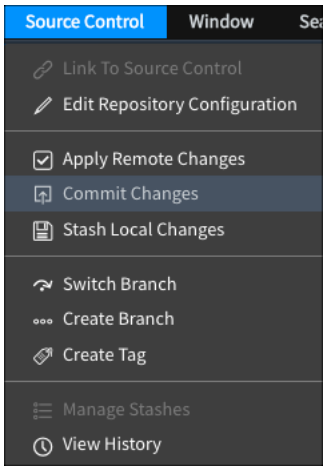
3.23 Exercise: Save Your Scripted REST API Work (Optional)

In this exercise, you will save the work you have done in this module to your GitHub repository.

> **NOTE**: For more details about connecting to GitHub, visit the **GitHub and ServiceNow section** (https://developer.servicenow.com/app.do#!/program/faq?q=github_and_servicenow) of the Developer Program FAQ.

Commit Changes

1. If the NeedIt application is not open in Studio from the last exercise, open it now.
    a. In the main ServiceNow browser window use the Application Navigator to open **System Applications > Studio**.
    b. In the Load Application dialog, click the **NeedIt** application.
2. Open the **Source Control** menu and select the **Commit Changes** menu item.



3. In the Commit Changes dialog, enter a Commit Message such as **Scripted REST API Exercises Completed**.
4. Click the **Commit Changes** button.
5. When the Commit Changes dialog reports success, click the **Close Dialog** button.

> **NOTE**: If you see a message from GitHub stating that you are unable to write to the repository, it means you have connected Studio to the ServiceNow version of the repository and not your forked version of the repository. The ServiceNow repository is read only. Your forked repository has read and write.

3.24 Scripted REST APIs Module Recap

Core concepts:

- Scripted REST APIs allow developers to create APIs to allow other applications to exchange information with their app
- Scripted REST APIs define:
    - Query parameters
    - API Documentation
    - Request headers
    - Response and request types

    - Resources

- Resources are defined in a Scripted REST API and consist of:
    - HTTP method (Relative path including path parameters
    - Resource path
    - Security
    - Allowed response format type
    - Resource documentation
    - Request header associations
    - Query parameter associations
    - Script

- In resource scripts:
    - The request and response objects are instantiated automatically.
    - The APIs for scripting resources are: RESTAPIRequest, RESTAPIRequestBody, RESTAPIResponse, and RESTAPIResponseStream
    - The process function is self-invoking
    - Populate the body object to return data to the application using the resource
    - Developers have access to the headers, query parameters, and path parameters

- Error objects help Scripted REST API consumers determine what went wrong
    - All error objects are created in the sn_ws_err namespace
    - Error objects can be pre-defined or custom
    - Error objects are returned in the response body and include an HTTP status code

- API versioning allows updating APIs without breaking existing integrations
    - Set a default version
    - Create documentation for each version of the API
    - Once enabled, versioning cannot be disabled
    - Version numbers are part of a the resource path

- The Usage by WEB API dashboard allows developers to view analytics for their APIs:
    - API Usage by Resource (Last 30 Days)
    - API Usage by Method (Daily)
    - REST API Usage by Version (Daily)
    - API Usage (Monthly)

- Use the REST API Explorer or a third party application (Postman, for example) to test Scripted REST APIs