knowledge15®
April 19–24, 2015 • Mandalay Bay • Las Vegas, NV

Lab Guide

Integration 201:
Consuming 3rd Party Web Services

Dragan Preocanin & Michael Starkey

Default Login / Password:

admin / Knowledge15

itil / Knowledge15

employee / Knowledge15

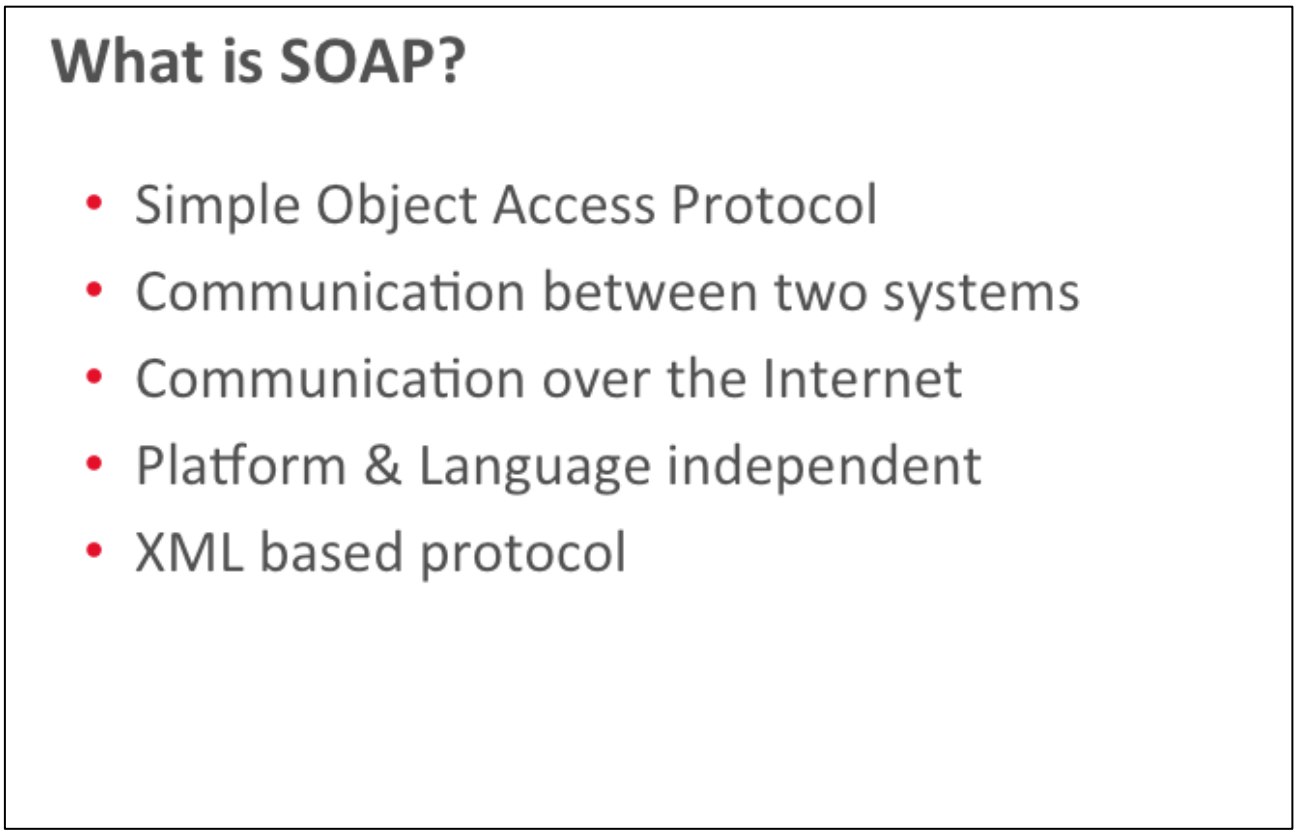
This

Page

Intentionally

Left

Blank



The SOAP protocol was designed to allow communication between multiple, unrelated systems. It was designed primarily for communication over a network such as the Internet.

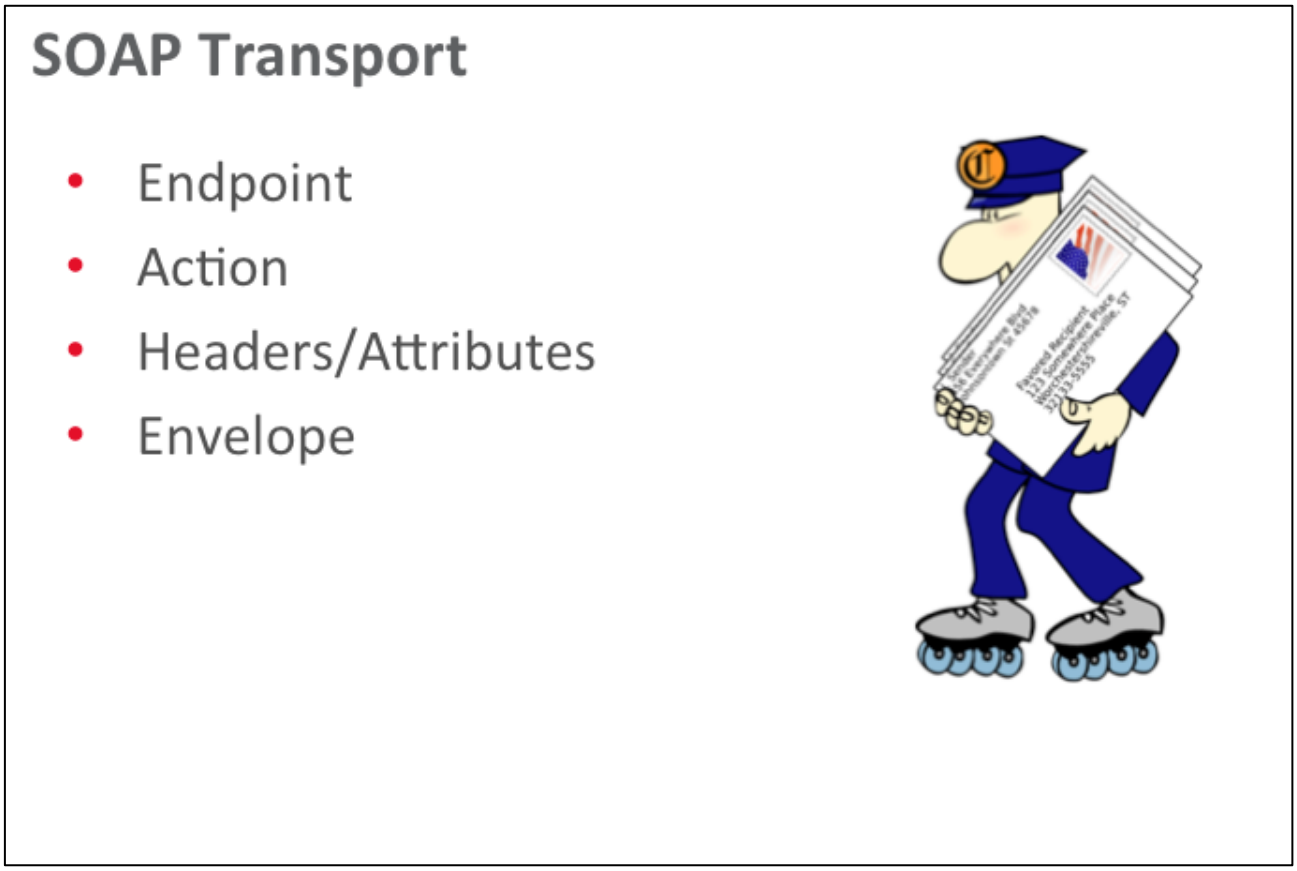
Working with SOAP eliminates concern about what language or platform a third-party service uses as long as the service can speak the SOAP protocol, you can interact with that third-party system without any knowledge of its implementation details.

The SOAP protocol is based on the XML markup language. It uses a series of XML documents to both describe the request, and provide a response.

®2015 ServiceNow, Inc. All rights reserved.

SOAP Transport

There are four main parts to a SOAP message transportation, depicted in the image below.



The **endpoint**. This is the URL or address where the SOAP request is delivered. It must be able to reach that endpoint over the network in order for the communication to succeed.

The SOAP **action** is a string that is often optional with the SOAP web service. However, some services do require this to be set. The string is often descriptive of what action should be performed by the web service.

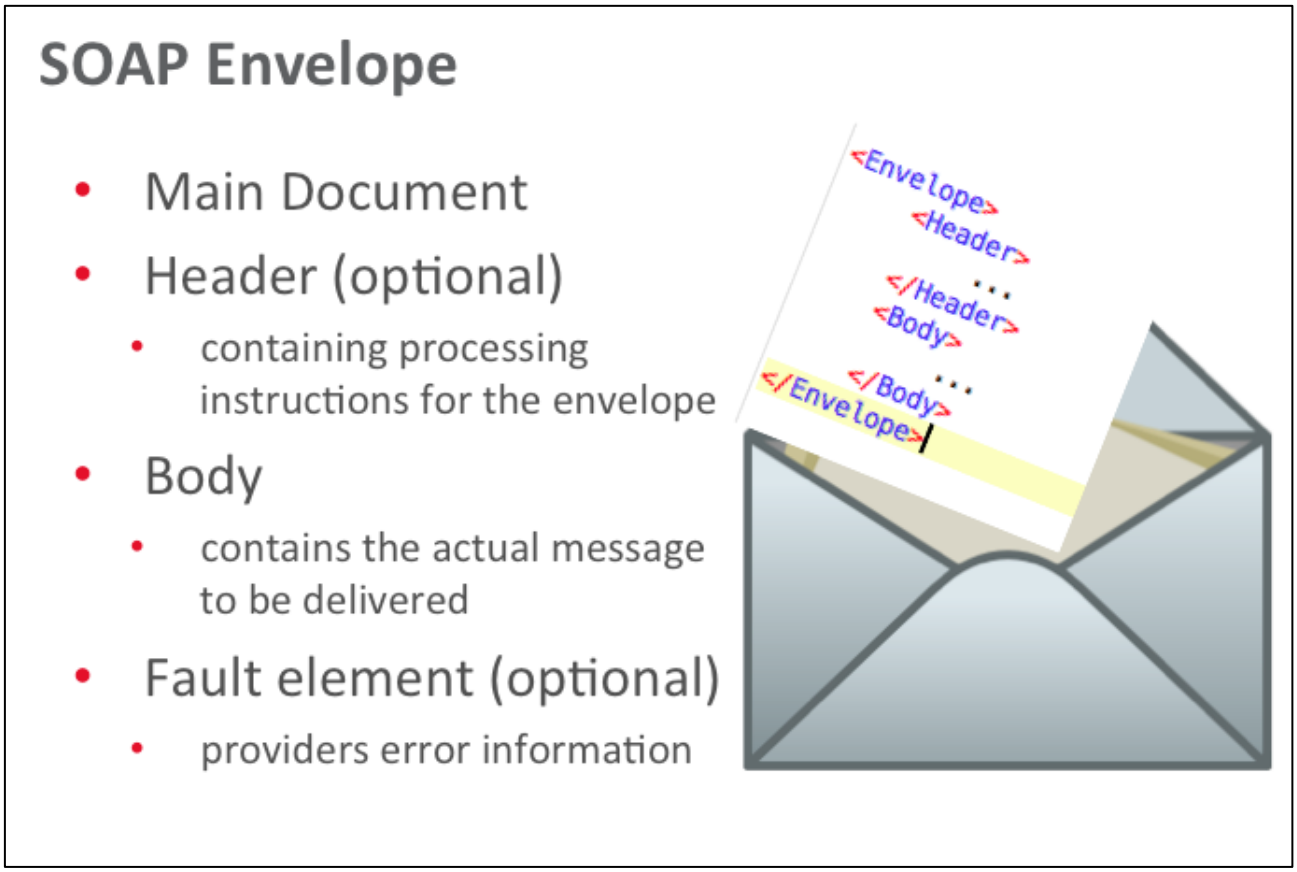
The **headers** and **attributes** of a message are found within the HTTP request. They contain information regarding the protocol being used as well as other custom information such as session IDs, authentication tokens, etc.

The **envelope** is the actual document that describes the details of the SOAP request. The envelope is written in XML format.

®2015 ServiceNow, Inc. All rights reserved.

SOAP Envelope

The SOAP **envelope** has a header and a body similar to an HTML webpage. Header and body details are dictated by the web service and explained in a corresponding WSDL document.



The **header** often contains processing instructions for the envelope, but it is not used in all web services. The web service documents any need for a header.

The **body** is the main part of the envelope, and is required for SOAP requests. It contains the actual message or request that is to be delivered to the web service.

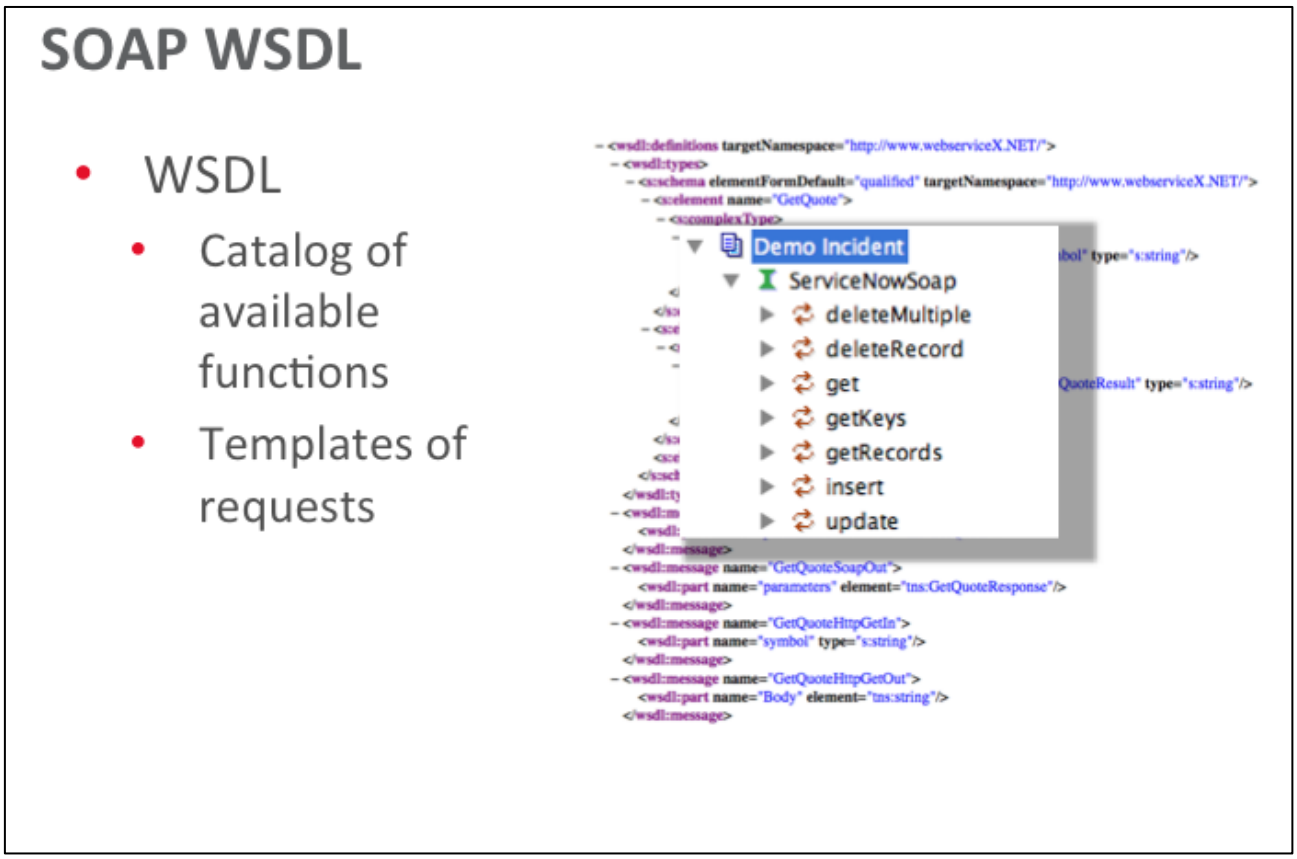
A third and final part of a SOAP envelope is the optional **fault element**. This element is often present if there was an error processing the web service request. The web service is responsible for populating this element with descriptive information to help the consumer understand what went wrong, and/or what to do next.

©2015 ServiceNow, Inc. All rights reserved.

SOAP WDSL

A **SOAP WSDL** is another XML document that is used in the SOAP protocol. A WSDL is a catalog of available functions provided by the web service.

A SOAP web service consuming client should be able to read the WSDL and generate a template of each of the available service requests provided by the web service.



©2015 ServiceNow, Inc. All rights reserved.

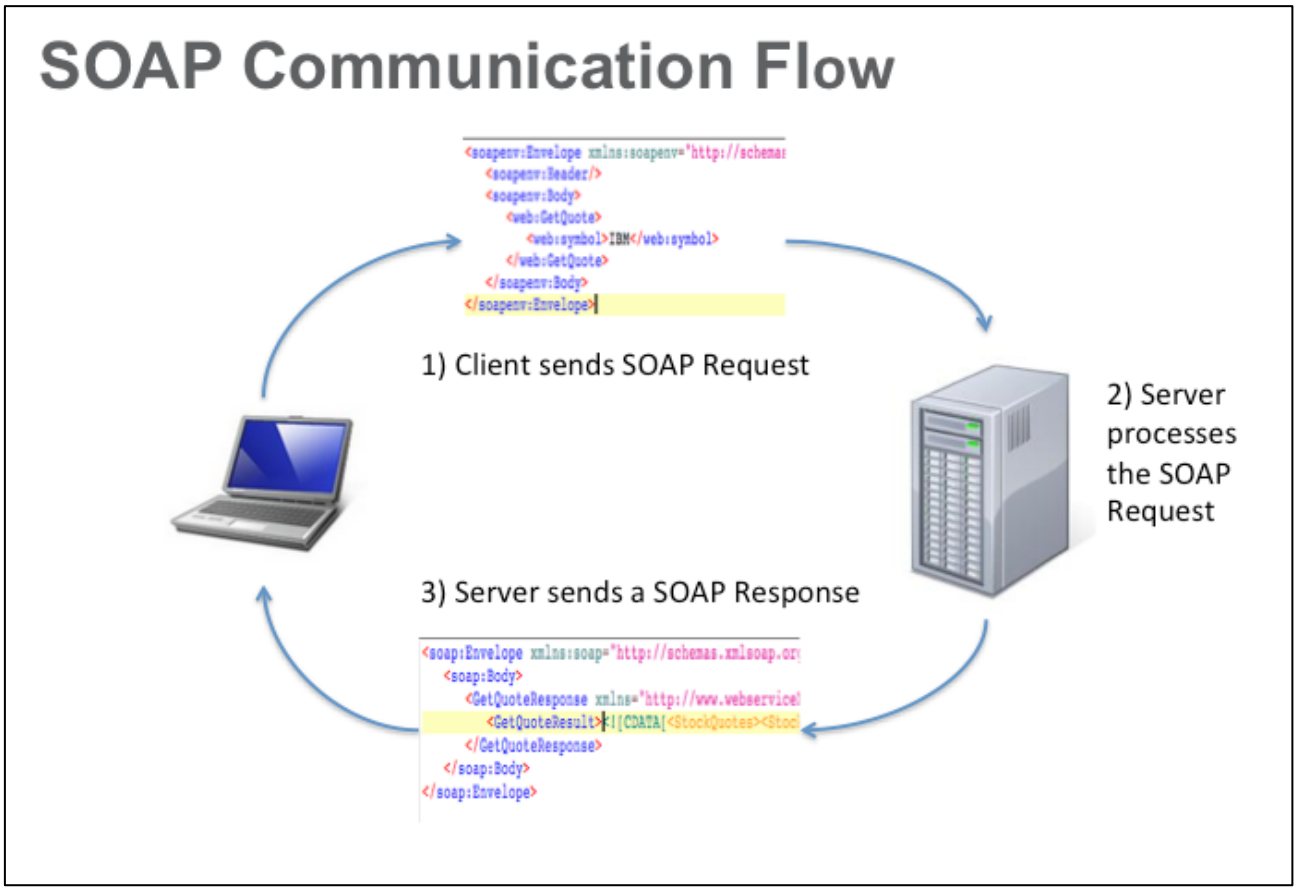
SOAP Communication Flow

There are three main parts of the standard **SOAP communication flow**.

Step 1 consists of a **client** that generates an XML envelope document and sends it to the service as a **SOAP request**.

Step 2 takes place when a server receives the SOAP request and processes that request in the manner in which it was coded.

Step 3 takes place when the server sends a SOAP response back to the client with an answer or response to the request that was submitted.



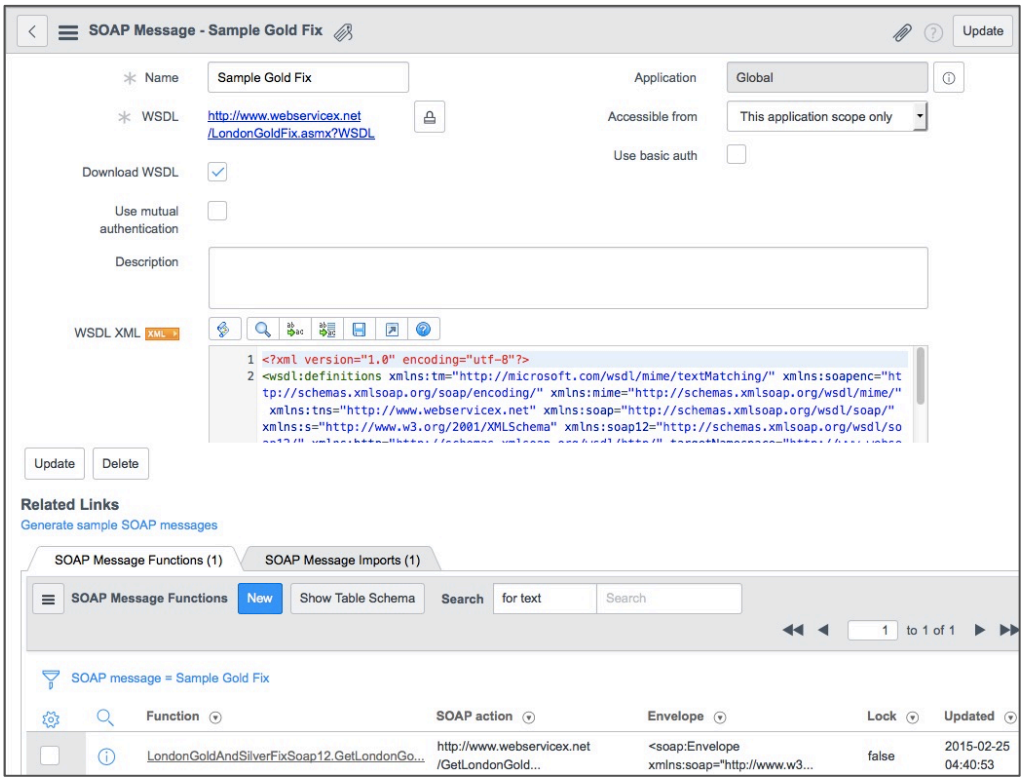
©2015 ServiceNow, Inc. All rights reserved.

SOAP Message UI

ServiceNow provides an easy to use web interface that allows you to build SOAP requests from a WSDL document.

With this tool you can easily specify the URL to a WSDL document and import it to generate the SOAP functions that are available in the web service. Alternatively, you can take a WSDL document and paste it directly into the tool to generate those functions if the WSDL is not available via a reachable URL.

Once you have the WSDL URL or WSDL document associated with your SOAP message record, you can click the generate sample SOAP messages link on the form to have ServiceNow generate templates for all of the available services provided by that SOAP web service.



®2015 ServiceNow, Inc. All rights reserved.

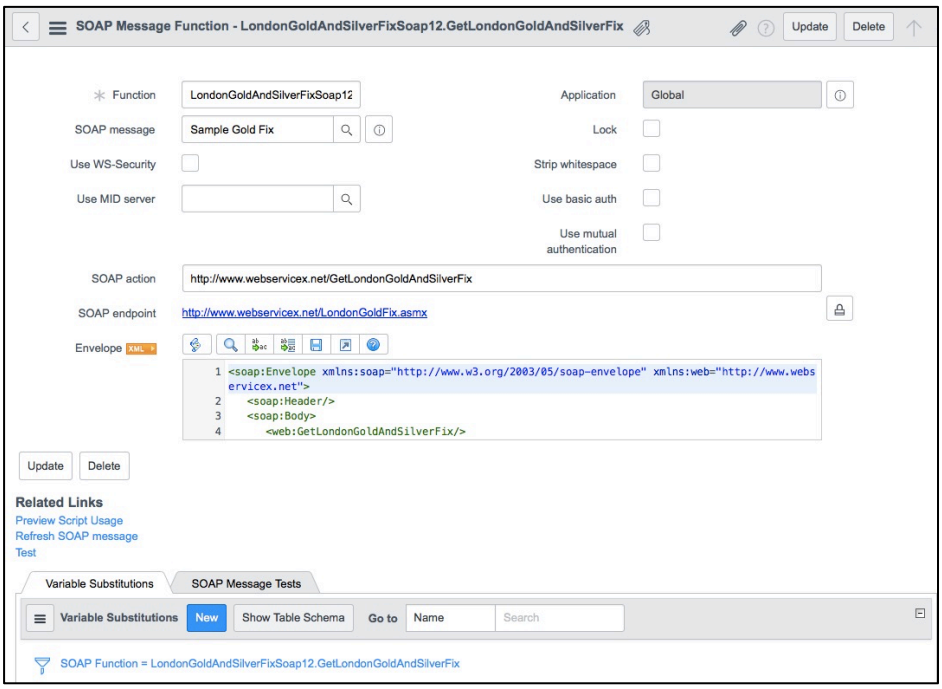
SOAP Message Function UI

After you generate some SOAP functions from the SOAP message record, you can open a function record to view the details on the template for the request.

Check the **Lock** field to ensure that future regenerations of SOAP functions do not overwrite the changes that you make to this function template.

ServiceNow supports the use of basic authentication, WS-security with outbound SOAP requests. You can select those check boxes to see how to set up those types of security.

Customize the SOAP action and/or endpoint manually by changing the text. With the SOAP endpoint, it may be necessary to click the padlock icon to change the link into a text box that can be edited.



The envelope document contains the XML template for the request. ServiceNow automatically detects possible elements where data will reside and creates variable strings in the form of **${VARIABLENAME}**. These can be used by scripts to make the SOAP request dynamic.

When using variables, you can create test values for those variables in the **OPEN** message parameters section of the form.

After you have the template set up the way that you want it, click the **test** link to submit the request to the web service. The tool shows you the response that is provided by the web service after the communication flow has finished.

©2015 ServiceNow, Inc. All rights reserved.

Lab Goal

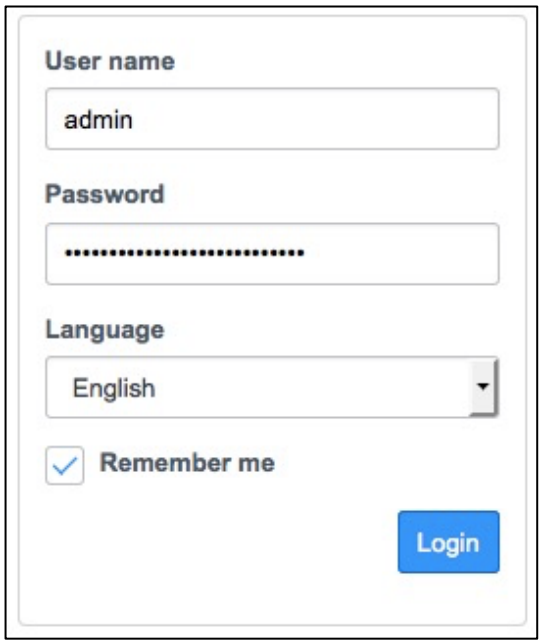
This lab provides a good understanding of how the SOAP Message GUI setup utility works within ServiceNow.

NOTE: Please see the 'Integration Lab 201 Knowledge' article in the 'Integration Lab 201' application for lab scripts and data that you can copy/paste in relevant steps to save you having to type these in manually.

Lab 1
SOAP Message

Log on to Your Training Instance

1. Login using the **admin** credentials provided on the cover.

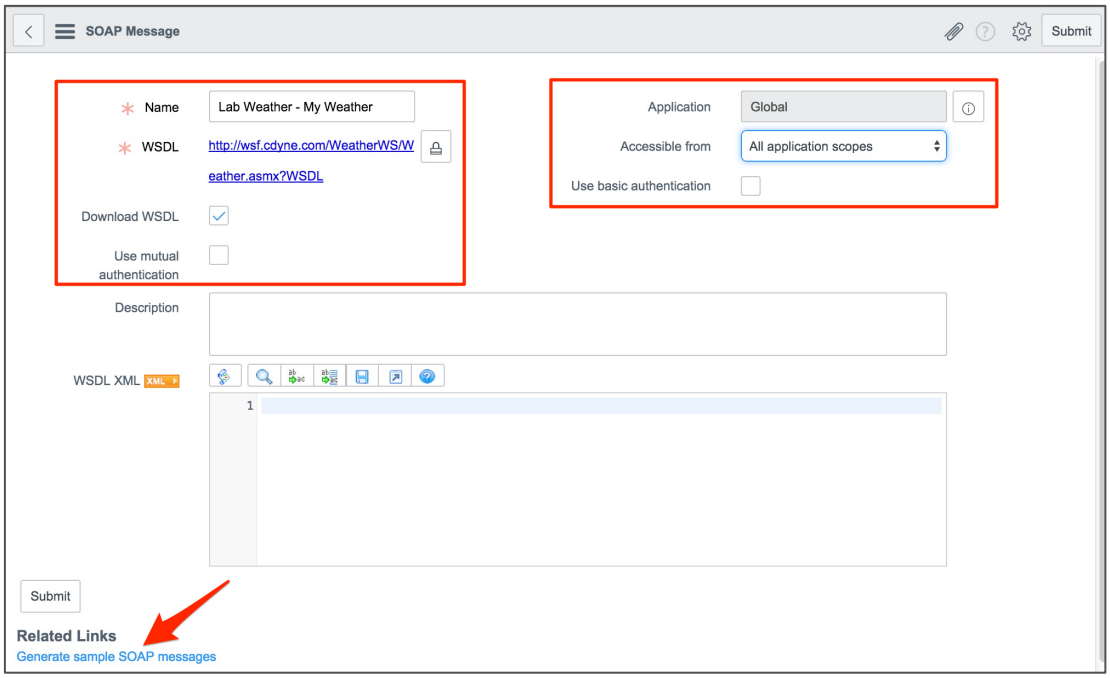


©2015 ServiceNow, Inc. All rights reserved.

Create a New SOAP Message Record

1. Navigate to System Web Services > Outbound > SOAP Message.
2. Click **New**.
3. Enter the following information into the fields:

   Name the record: **Lab Weather – My Weather**
   Application: **Global**
   Accessible from: All application scopes
4. Confirm that the Download WSDL check box is checked.
5. **Ent**er the follo**wing WSDL** URL (you may have to click the padlock icon first): http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL.
6. Click Generate sample SOAP messages.

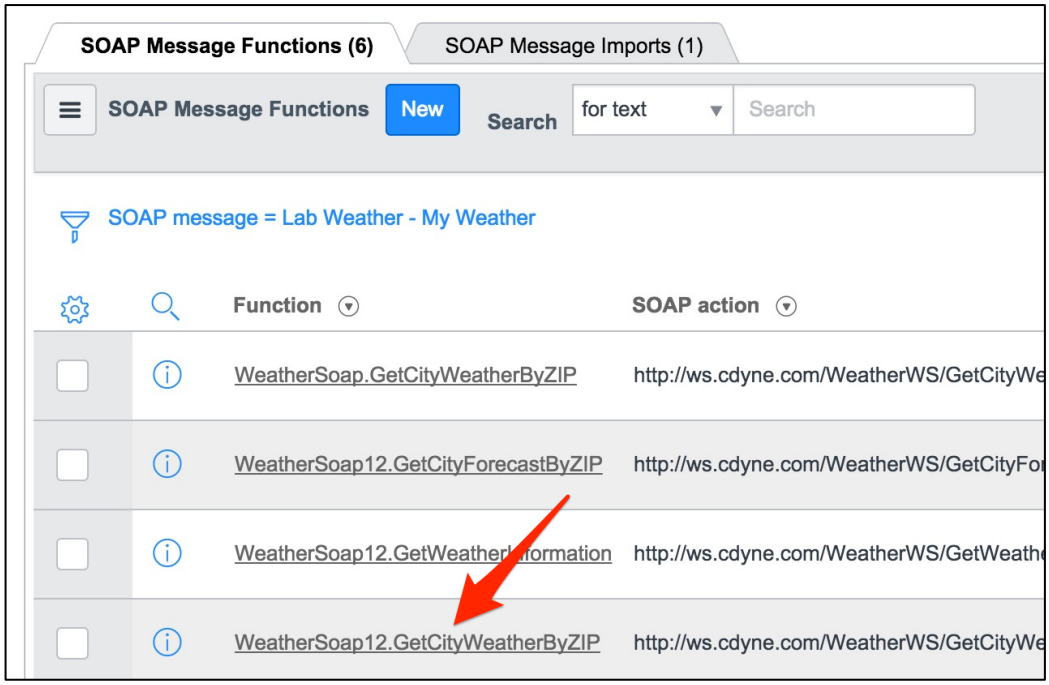


7. If there is a Warning, click **OK**.

©2015 ServiceNow, Inc. All rights reserved.

Set Up a SOAP Message Function

1. Navigate to **System Web Services > Outbound > SOAP Message**.
2. Open the **Lab Weather – My Weather** SOAP Message record.
3. Open the following SOAP function: **WeatherSOAP12.GetCityWeatherByZIP**.



4. **Note**: The **variable name** from the auto generated SOAP Envelope. SOAP function variables are in the format of **${VAR_NAME}**.



©2015 ServiceNow, Inc. All rights reserved.

5. Scroll to the **Variable Substitutions** related list on the form**.** Click **New** and set the following fields:

   Name: **ZIP**
   Escape Type: **No escaping**
   Test Value: **99019**
   Application: **Global**

6. Click **Submit**.

7. Confirm that the new variable and value are listed.



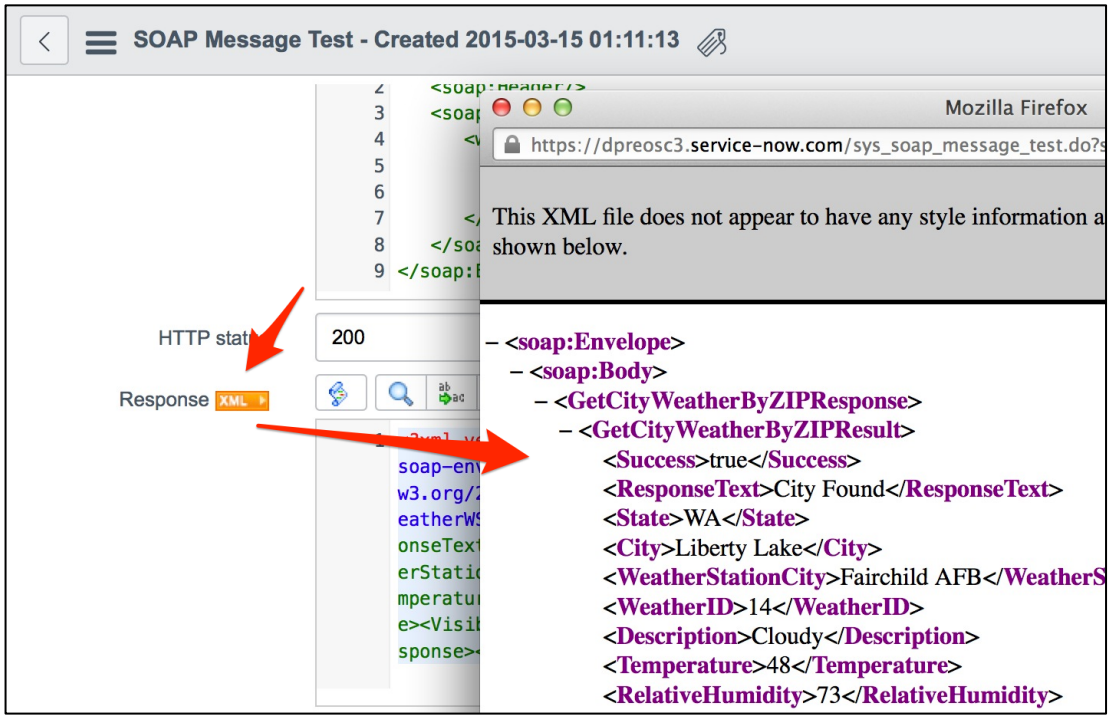
8. Click the **Test** link on the SOAP Message Function form to test the web service. The HTTP status of the response should have a value of **200**.

9. You should see a screen that lists the actual **Request** that was sent to the server. Notice that the **${ZIP}** string no longer exists in the XML payload, but has been replaced by the value you set in the prior step.



©2015 ServiceNow, Inc. All rights reserved.

10. Click the XML icon on the **Response** field to get a formatted view of the SOAP Response.



Report Your Completion of This Lab Exercise

1. Navigate to **Lab Management > Report Lab Progress**.

2. Enter your progress status.

©2015 ServiceNow, Inc. All rights reserved.

Scripting with SOAP

After you have set up your SOAP message function templates, you can leverage them within ServiceNow scripts.

Some of the most common scripting areas where SOAP messages are called from include business rules, script includes, workflows, and other server-side scripts.

There are a few script include libraries that support a user in sending out SOAP requests:

The **SOAPMessage** library is the API around the GUI tool that we have just demonstrated. Use it to set and point values, set variables, submit the request, etc.

The **SOAPEnvelope** and **SOAPRequest** libraries are leveraged under the covers by the SOAP Message tool but can also be used by an administrator in creating customs SOAP messages, bypassing the GUI SOAP Message utility, if necessary.



©2015 ServiceNow, Inc. All rights reserved.

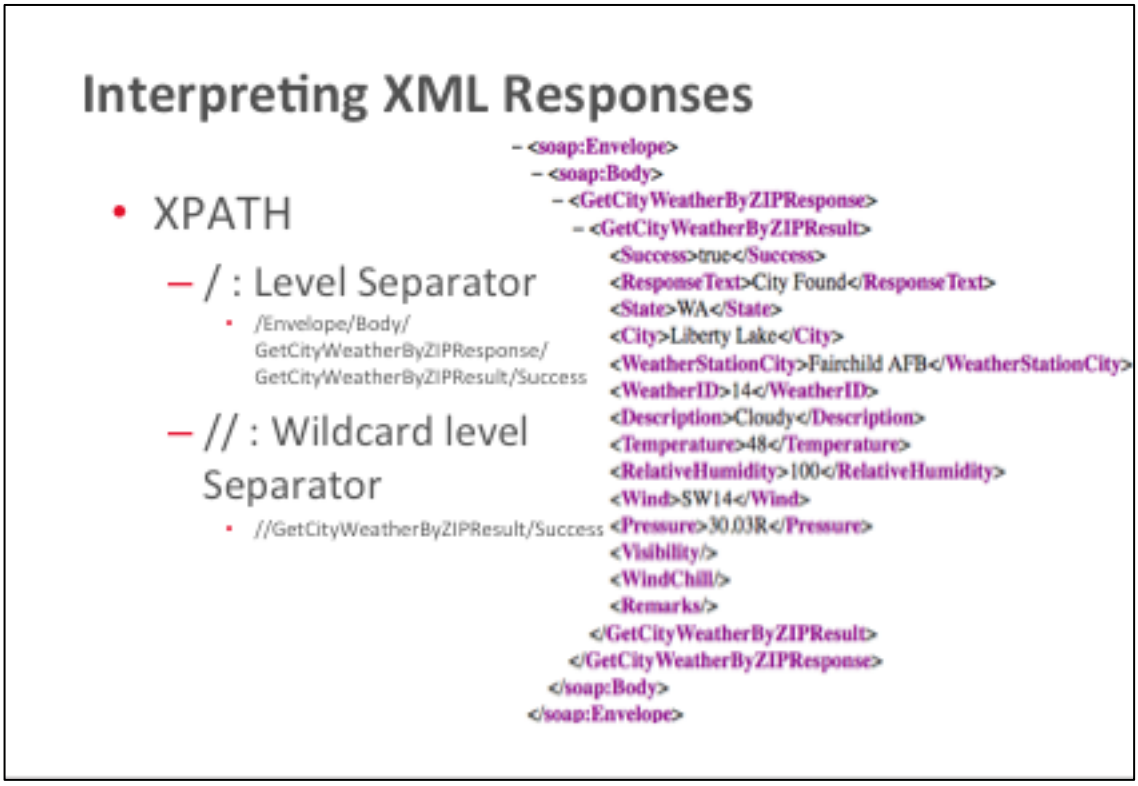
Interpreting XML Responses

XPATH is a standard mechanism used to parse XML documents. ServiceNow supports the use of XPATH in interpreting the SOAP responses.

While XPATH contains a number of syntax and functions available to it to make it a very powerful tool in interpreting XML documents, the most basic and primary element that you will typically have to be familiar with is the "**/**" character. This character is a level separator similar to Linux/UNIX paths on a file system.

The other common character that is used is a "**//**" character. This character denotes a relative path, rather than a strict path.

Another widely used character is the "*****" character. It is denoted as a wildcard character in a string.

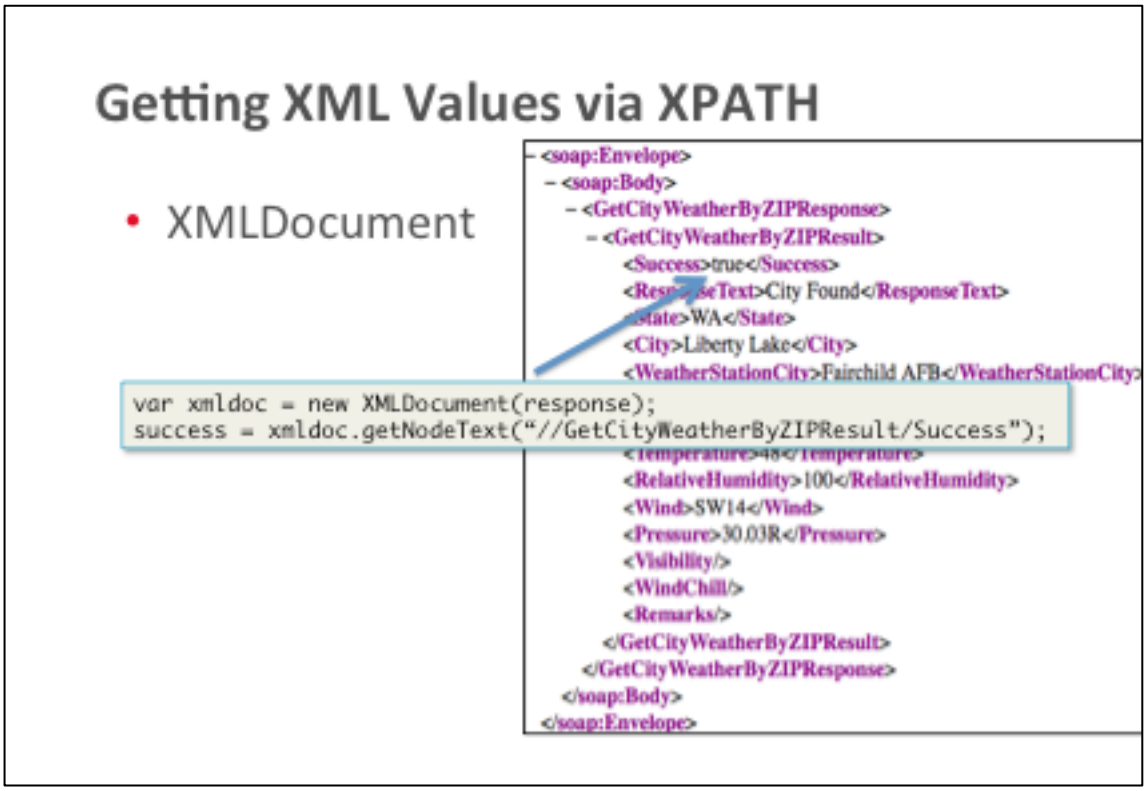


©2015 ServiceNow, Inc. All rights reserved.

Getting XML Values via XPATH

ServiceNow has a script include that facilitates operations that you can perform on XML documents. This library is called **XMLDocument**.

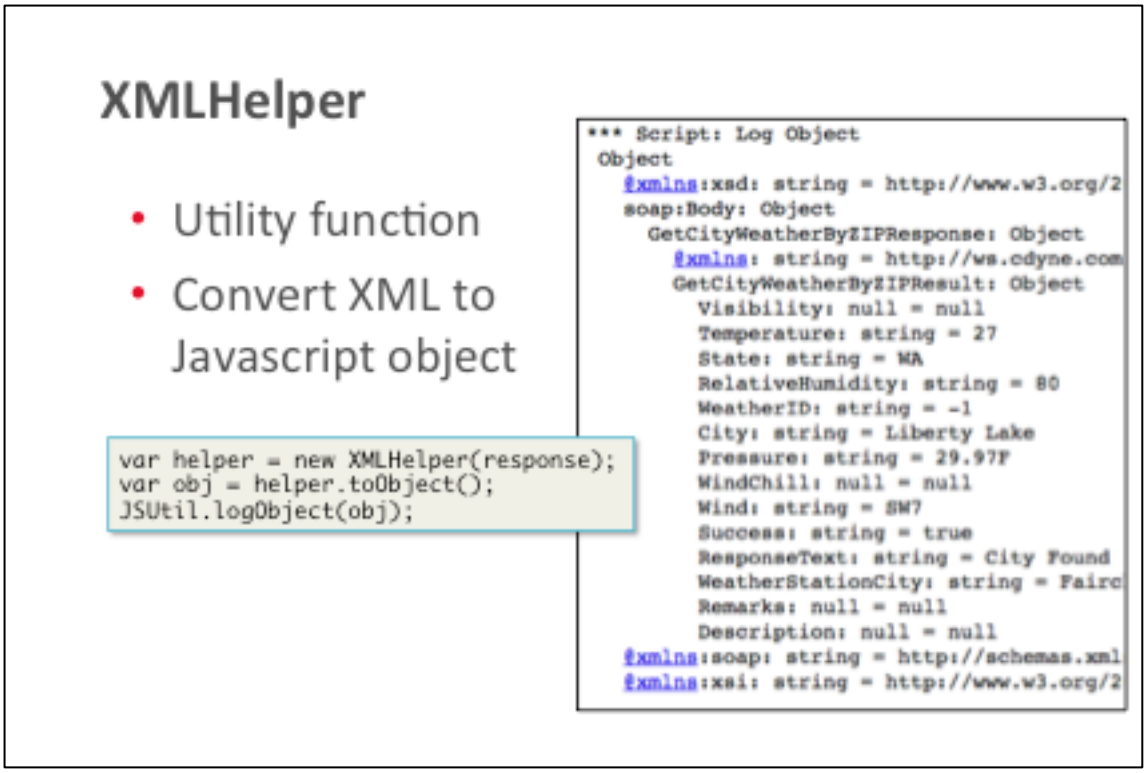
With this library, you can load an XML string and perform operations on it. A common operation that you will perform is the **getNodeText** function that takes an **XPATH** string in and attempts to parse the XML document with the XPATH to find the element that you are interested in and return the contents of that element node.



©2015 ServiceNow, Inc. All rights reserved.

XML Helper

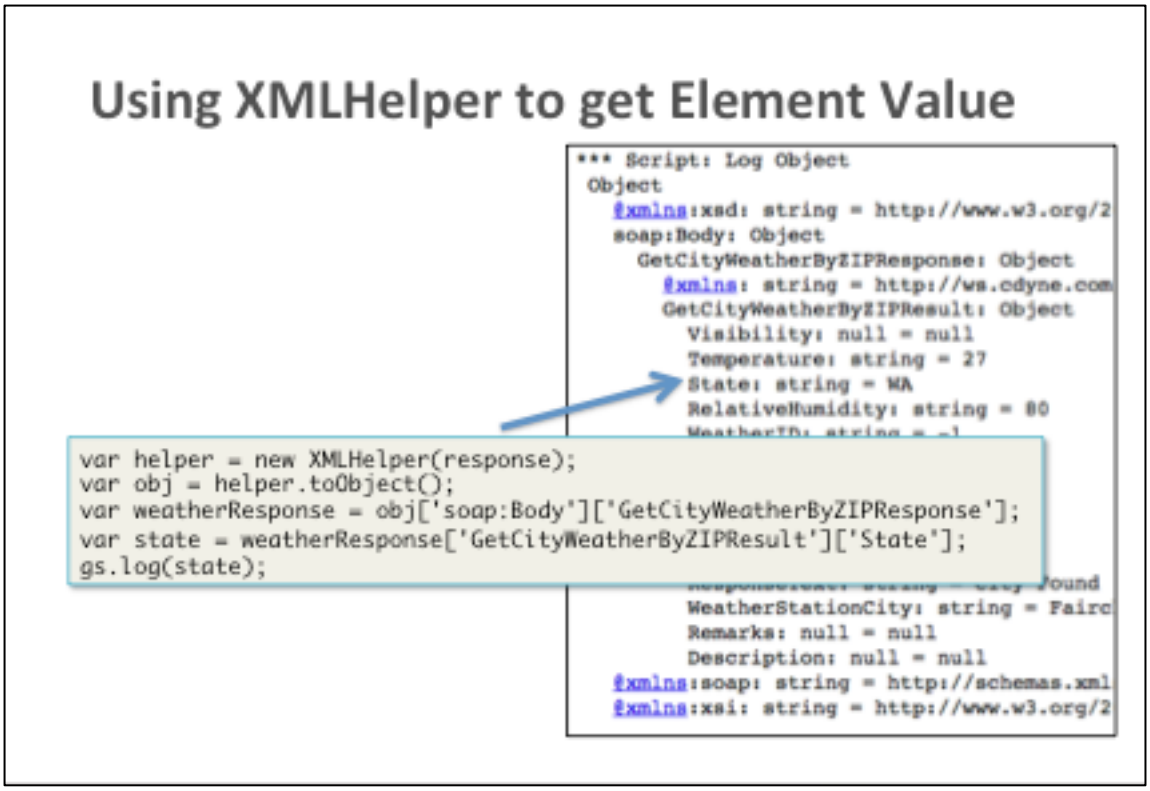
Another handy function that is provided by ServiceNow is the **XMLHelper** library. This library has a number of functions, however, a very commonly used function is the one that will take an XML string and convert it to a JavaScript object.



®2015 ServiceNow, Inc. All rights reserved.

Using XMLHelper to Get Element Value

XMLHelper allows you to access the parts of the XML document as if the various elements were member variables to an object. Each level down the XML document is represented by member variables that are new objects themselves.



©2015 ServiceNow, Inc. All rights reserved.

Lab Goal

This lab explains how to leverage SOAP Message through server-side Scripts.

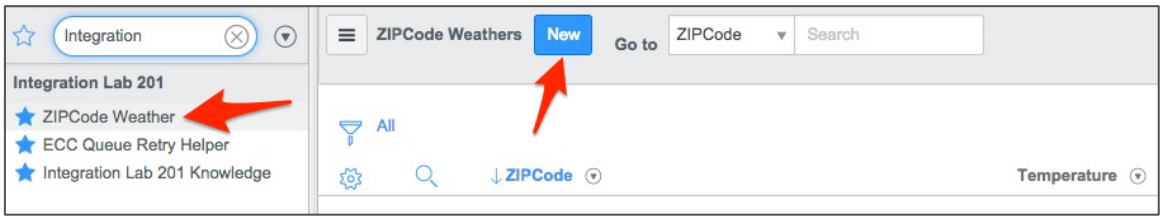
In this exercise, you create a business rule on the **ZIPCodeWeather** table to update the weather for a ZIPCode any time it is created or modified.

NOTE: Please see the 'Integration Lab 201 Knowledge' article in the 'Integration Lab 201' application for lab scripts and data that you can copy/paste in relevant steps to save you having to type these in manually.

Lab 2
Scripted SOAP Message

Browse to the ZIPCode Weather Module

1. Browse to the **Integration 201 Lab** application in ServiceNow.
2. Click on the **ZIPCode Weather** module.
3. Click the **New** button to create a new record.

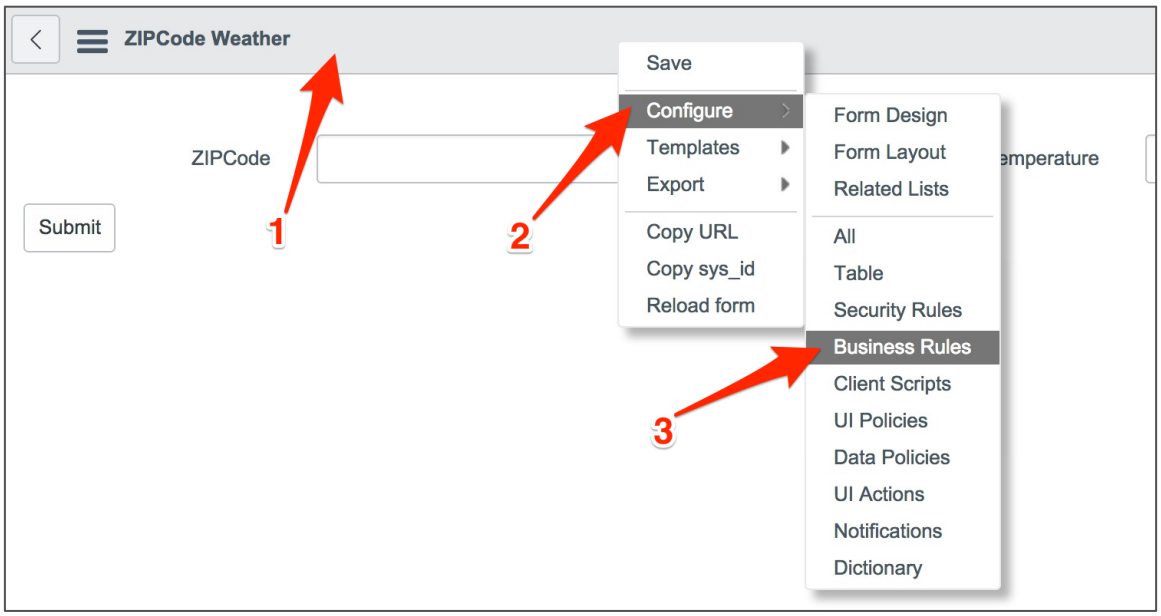


Create a New Business Rule on the Form

1. Right click the **ZIPCode Weather** title bar.
2. Select **Configure > Business Rules** in the popup menu.

©2015 ServiceNow, Inc. All rights reserved.

3. Click **New** to create a new Business Rule.



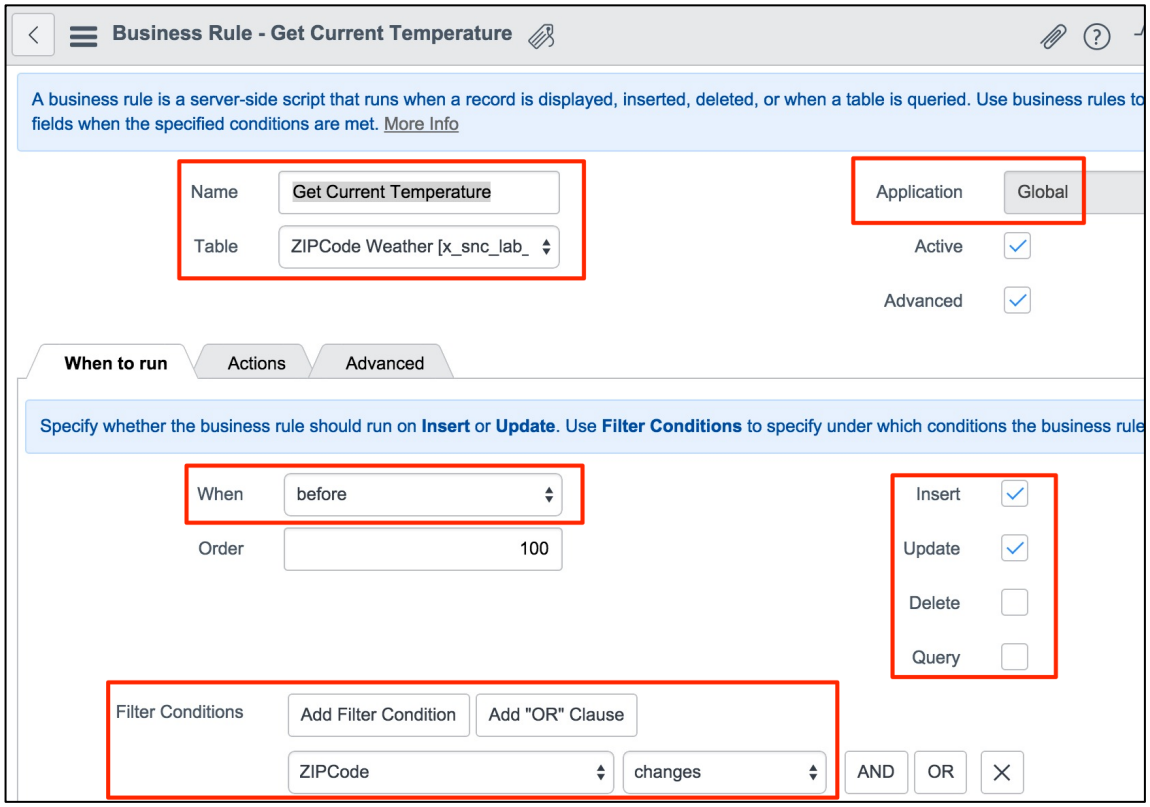
Set Up the Business Rule Parameters

1. Fill out the business rule form with the following information.

Name	Get Current Temperature
Table	ZIPCode Weather
Application	Global
Advanced	Checked
When	before
Insert	Checked
Update	Checked
Condition	Filter: ZIPCode: changes

©2015 ServiceNow, Inc. All rights reserved.

2. Review the form, which should look like the example.



3. Click **Submit**.

Browse to the SOAP function

1. Browse to **System Web Services > Outbound > SOAP Message**.
2. Select the **Lab Weather – My Weather** SOAP Message record.
3. In the SOAP Message Functions list, select the **WeatherSOAP12.GetCityWeatherByZIP** function.

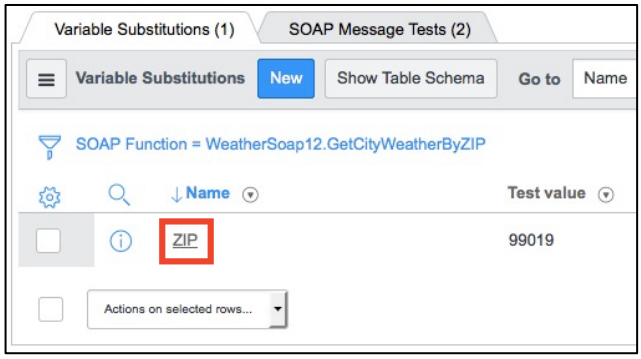
©2015 ServiceNow, Inc. All rights reserved.

Generate a Sample SOAP script

1. Ensure that there is a **ZIP** variable setup in the SOAP envelope.



2. Ensure that there is a **ZIP** variable created in the **SOAP Message Parameters** list.

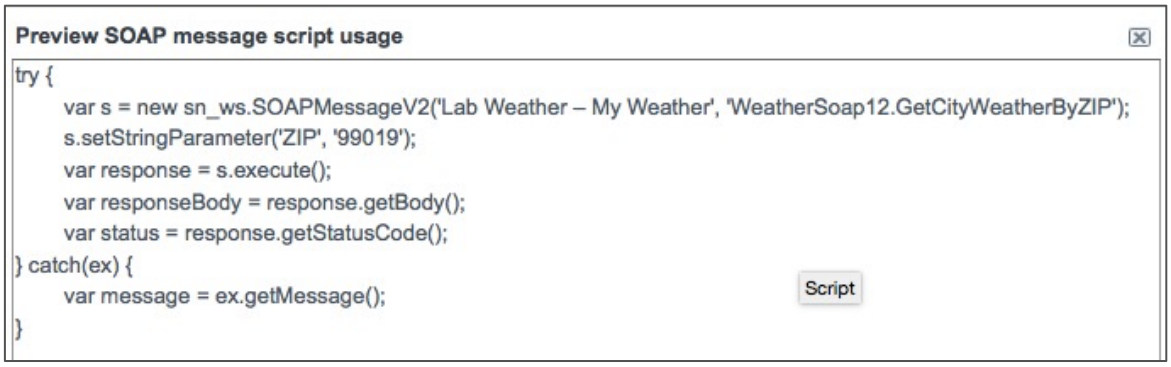


3. Click **Preview script usage**.



©2015 ServiceNow, Inc. All rights reserved.

4. Copy the code that is generated in the popup dialog.

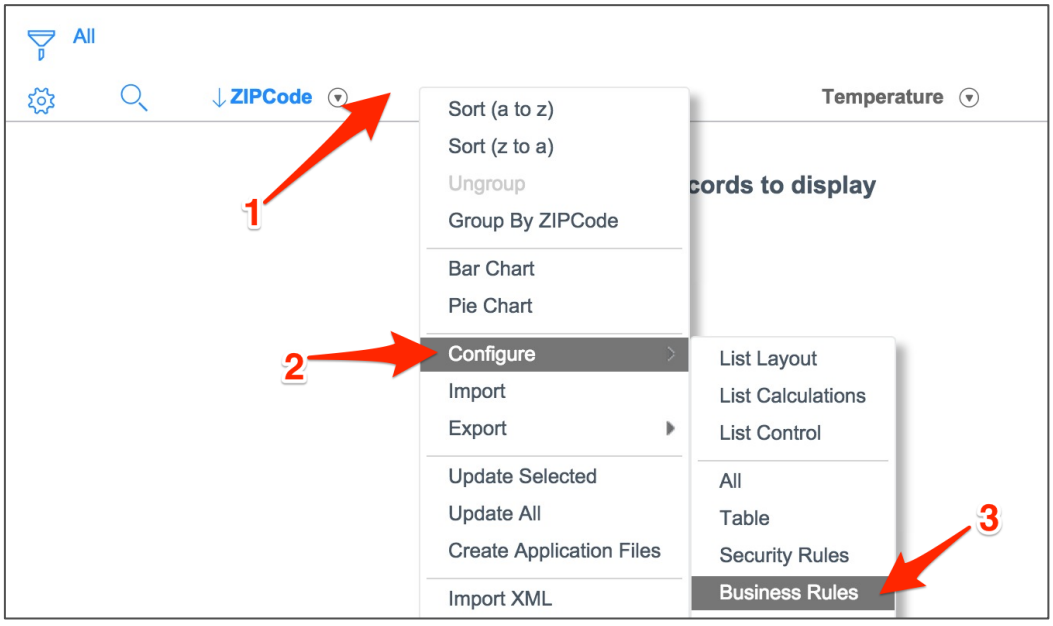


5. Close the preview dialog window.

Paste and Modify the Script in the Business Rule

1. Navigate to **Integration Lab 201 > ZIPCode Weather**.

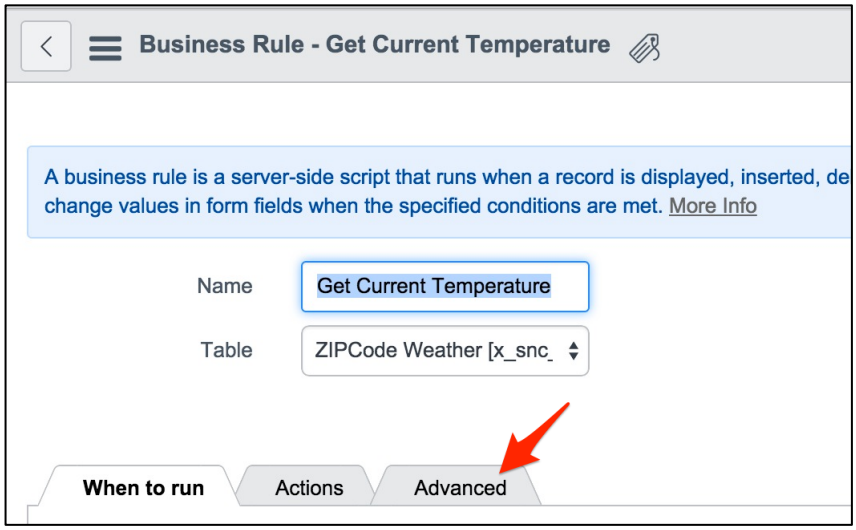
2. Right-click on the list title bar and choose **Configure > Business Rules** from the menu options.



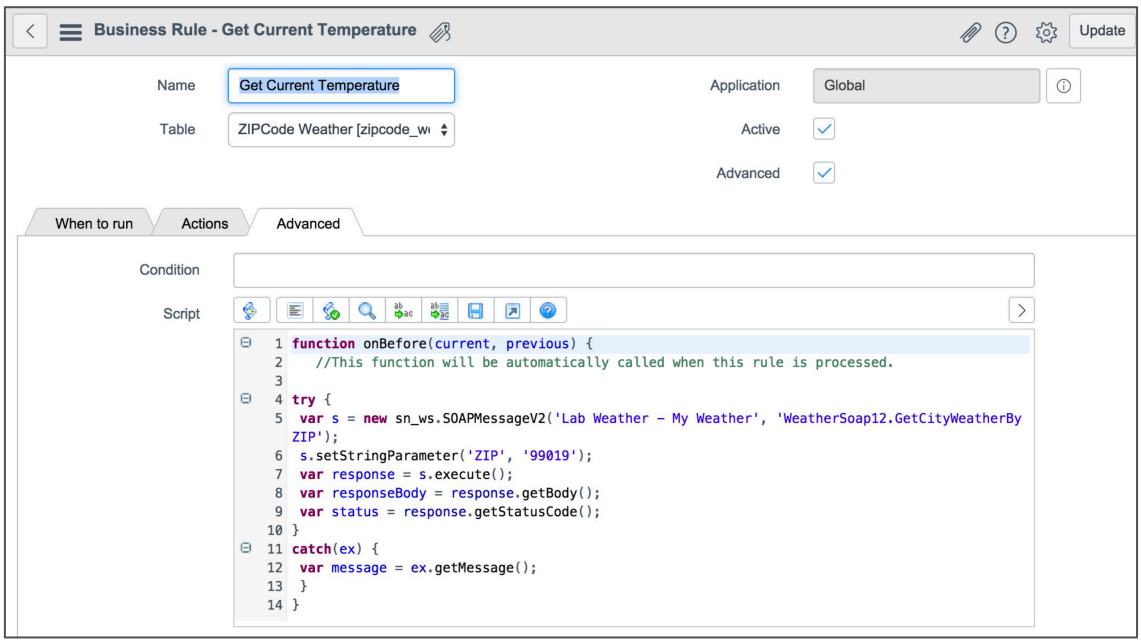
3. Open the **Get Current Temperature** business rule record.

©2015 ServiceNow, Inc. All rights reserved.

4. Click the **Advanced tab** on the business rule form.



5. Paste your SOAP script in the **script** field.



®2015 ServiceNow, Inc. All rights reserved.

6. Instead of hard coding the zip code in the **setStringParameter** function, change the **99019** to be the variable: **current.zipcode.**

```
1  function onBefore(current, previous) {
2     //This function will be automatically called when this rule is processed.
3
4  try {
5   var s = new sn_ws.SOAPMessageV2('Lab Weather - My Weather', 'WeatherSoap12.GetCityWeatherBy
   ZIP');
6   s.setStringParameter('ZIP', current.zipcode);
7   var response = s.execute();
8   var responseBody = response.getBody();
9   var status = response.getStatusCode();
10 }
11 catch(ex) {
12   var message = ex.getMessage();
13   }
14 }
```

Set the Temperature Value on the Form

1. Add the following lines to the script.

```
//Convert the XML response string to an XMLDocument object
var xmldoc = new XMLDocument2();
xmldoc.parseXML(responseBody);
current.temperature = xmldoc.getNodeText("//Temperature");
```

The script should now look like the following:

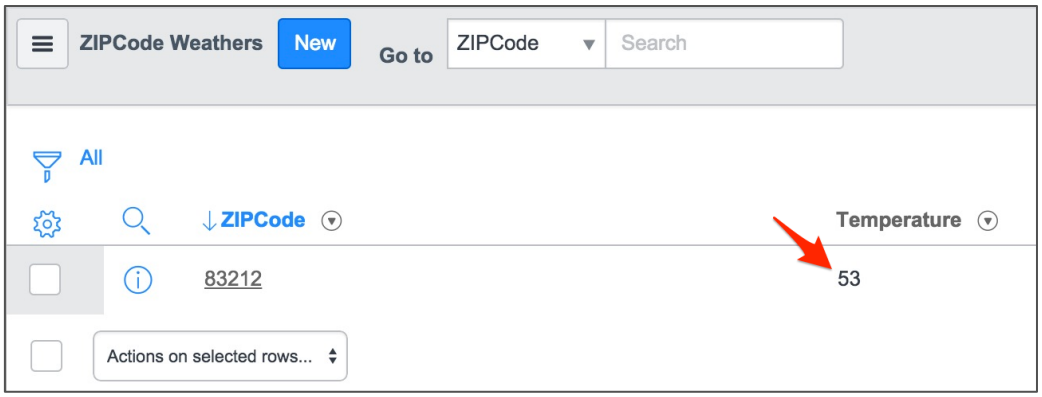
```
1  function onBefore(current, previous) {
2     //This function will be automatically called when this rule is processed.
3  try {
4  var s = new sn_ws.SOAPMessageV2('Lab Weather - My Weather', 'WeatherSoap.GetCityWeatherByZIP
   ');
5   s.setStringParameter('ZIP', current.zipcode);
6   var response = s.execute();
7   var responseBody = response.getBody();
8   var status = response.getStatusCode();
9
10  //Convert the XML response string to an XMLDocument object
11  var xmldoc = new XMLDocument2();
12  xmldoc.parseXML(responseBody);
13  current.temperature = xmldoc.getNodeText("//Temperature");
14
15 }
16 catch(ex) {
17   var message = ex.getMessage();
18   }
19 }
```

2. Click **Update**.

©2015 ServiceNow, Inc. All rights reserved.

Test the Scenario

1. Navigate to **Integration Lab 201 > ZIPCode Weather**.
2. Click **New.**
3. Enter a **ZIPCode** value of **83212.**
4. Click **Submit**.
5. Confirm that the temperature has been set on the record automatically.



Report Your Completion of This Lab Exercise

1. Navigate **to Lab Management > Report Lab Progress**.
2. Enter your progress status.

©2015 ServiceNow, Inc. All rights reserved.

What is REST?

Like SOAP, **REST** is a protocol for requesting information from a server. However, it is lightweight in comparison, it does not need to use XML and supply the required XML structure each time it needs to make a request like SOAP and in most cases it relies on a simple URL. REST-based web services can provide a response in a number of formats like JSON, CSV as well as XML.

The ServiceNows outbound REST functionality allows you to retrieve, create, update, or delete data on a web server that supports the REST architecture. You can send REST messages either directly or from a MID Server, which allows REST to access an endpoint within the internal network. REST messages can be sent using scripts such as Business Rules or a workflow activity to retrieve or manage data on a REST endpoint and the returned response can then be used in the ServiceNow instance.

REST supports the following HTTP Methods:

GET
POST
PUT
DELETE



©2015 ServiceNow, Inc. All rights reserved.

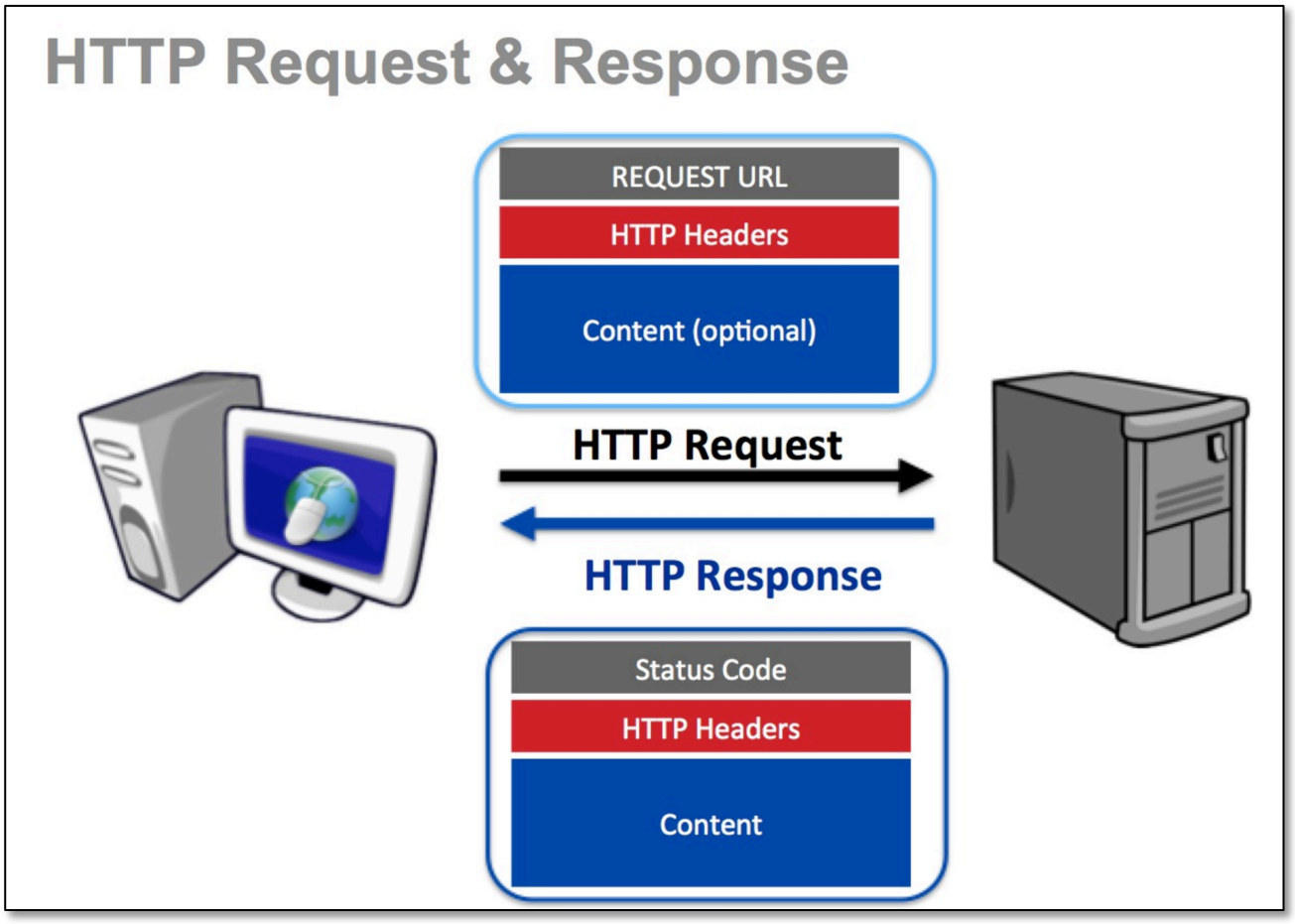
HTTP Request and Response

Request URL – Specifies the address of the resource + the HTTP verb identifying the desired operation (e.g., GET, PUT, POST or DELETE).

Query Parameters are optional and appended to the URL path of the request.

HTTP Headers are a required component of the message and contain metadata for the transaction (e.g. encoding, date, authorization).

Content or **Body** is optional and contains data to be transmitted.



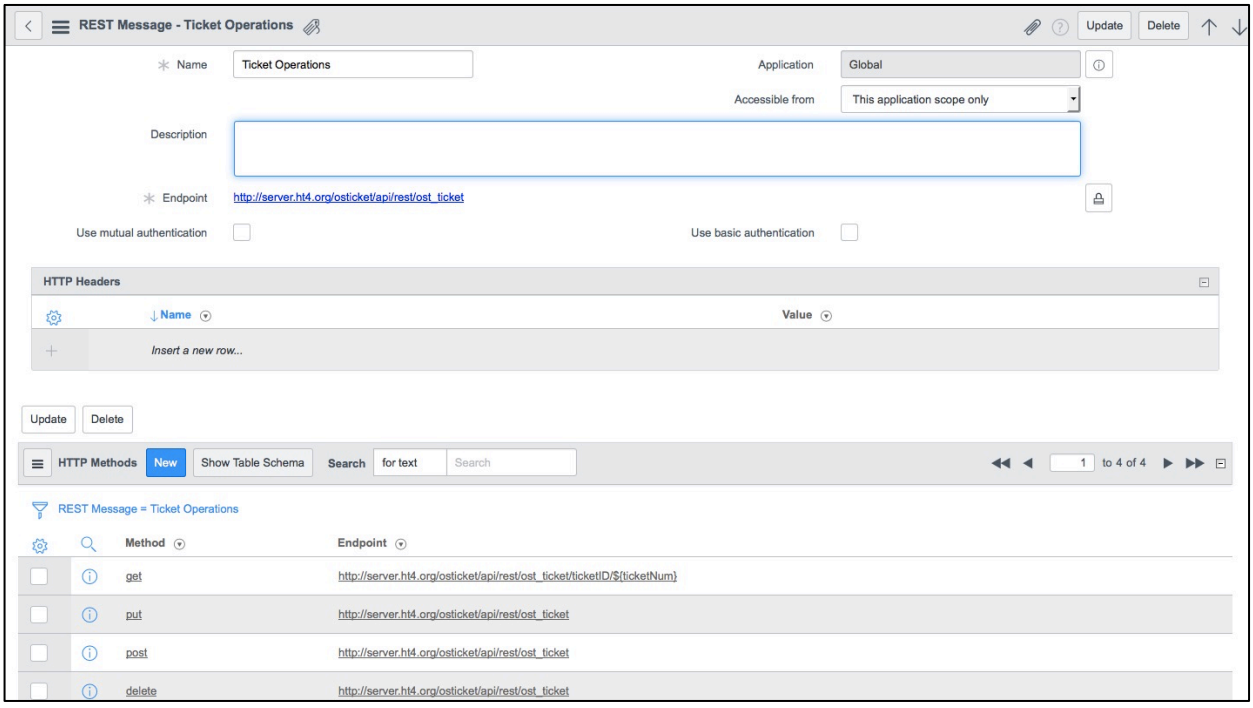
©2015 ServiceNow, Inc. All rights reserved.

Making Outbound REST Calls from ServiceNow

- GUI based tool – **REST Message**
- Similar to the **SOAP Message** tool
- Requires an **endpoint**
- Generates the four HTTP Methods as **functions**
- Endpoint can use variables
  - **${varName}**

REST Message UI

The example depicts a REST message UI.



®2015 ServiceNow, Inc. All rights reserved.

REST Function Options

- Variables can be used in various parts of the function
  - Endpoint URL
  - Header names or values
  - Function parameter definition names or values
  - Content
- Connectivity options
  - Authentication
  - MID Server
- Access to HTTP request components
  - Headers
  - Parameters
  - Content (POST, PUT only)
- Set variable values
  - String vs. XML
- Test the request
- Preview script usage

©2015 ServiceNow, Inc. All rights reserved.

Lab Goal

Lab 3
REST
Message

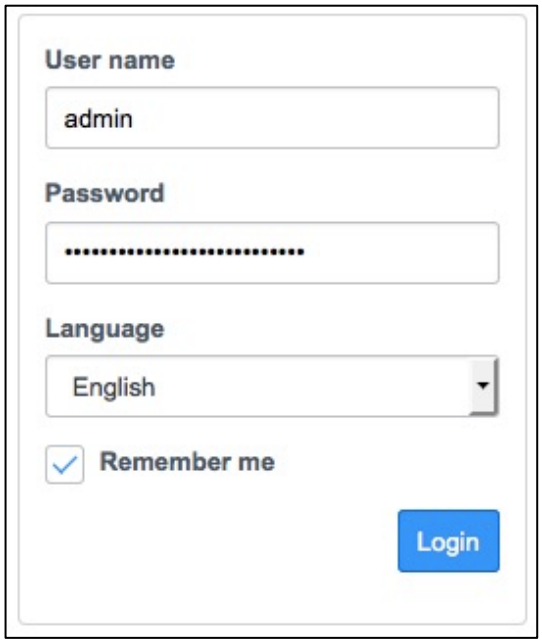
This lab explains how to submit REST based requests using the REST Message feature in ServiceNow.

In this exercise, you leverage the GUI REST Message utility to make REST-based requests to a third party ticketing system to create a ticket and retrieve its information.

NOTE: Please see the 'Integration Lab 201 Knowledge' article in the 'Integration Lab 201' application for lab scripts and data that you can copy/paste in relevant steps to save you having to type these in manually.

Log on to Your Training Instance

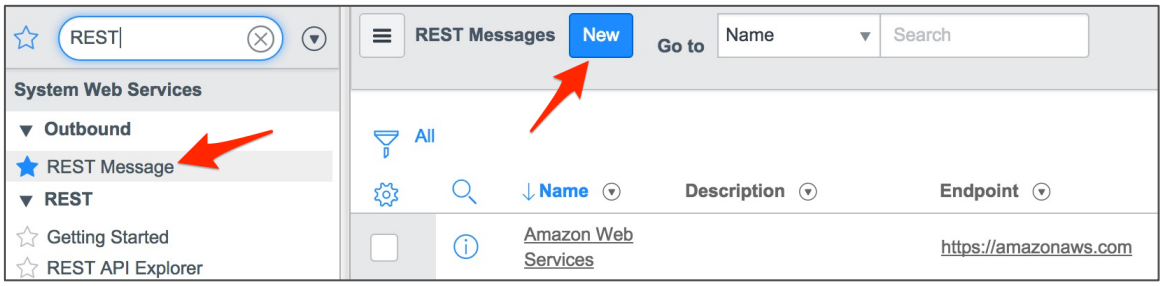
1. Login using the **admin** credentials provided on the cover.



©2015 ServiceNow, Inc. All rights reserved.

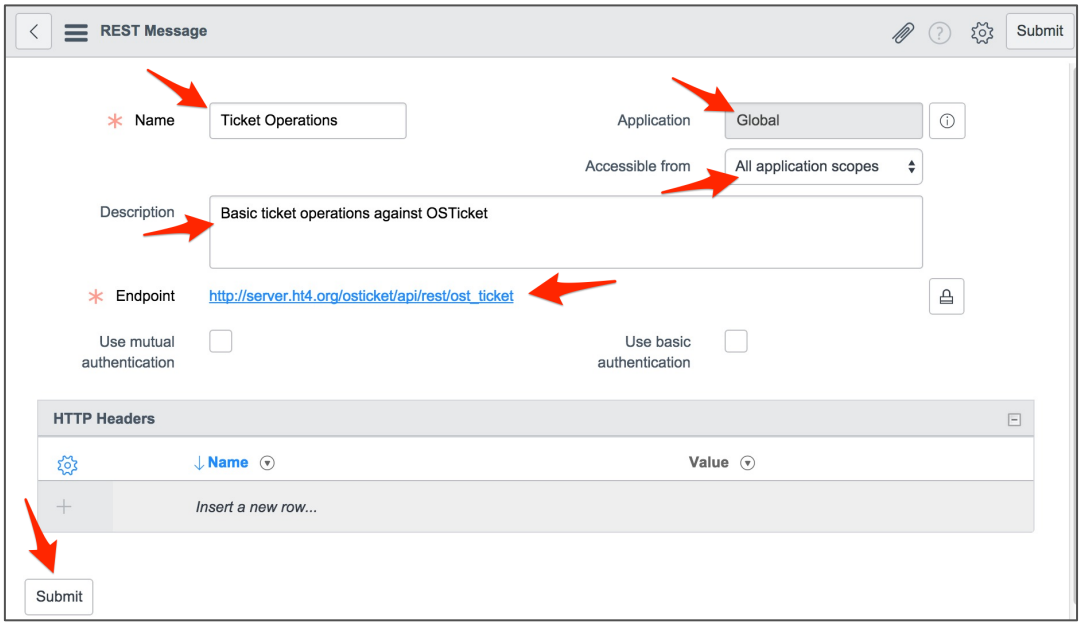
Configure REST Message

1. In the ServiceNow instance, navigate to **System Web Services > Outbound > REST Message** and click **New**.



2. Fill in the following values for the fields:

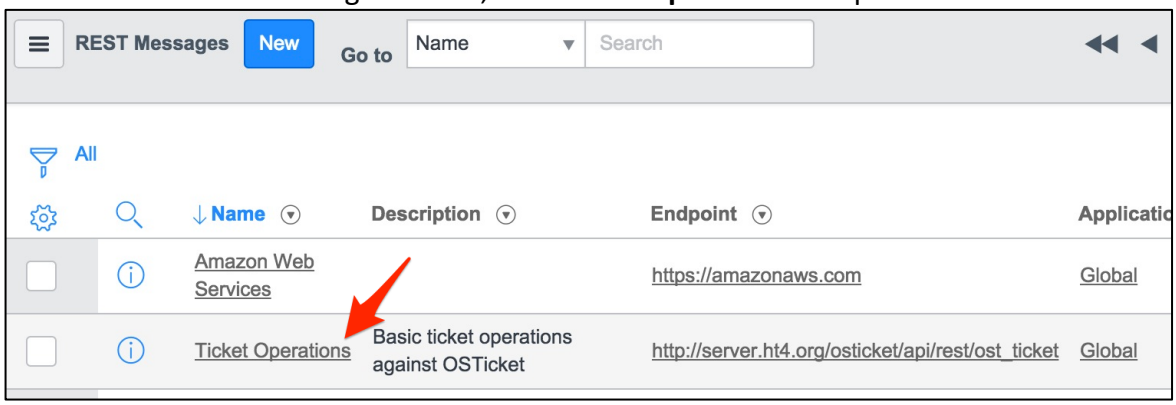
Name	**Ticket Operations**
Description	**Basic ticket operations against OSTicket**
REST Endpoint	**http://server.ht4.org/osticket/api/rest/ost_ticket**
Application	**Global**
Accessible from	**All application scopes**



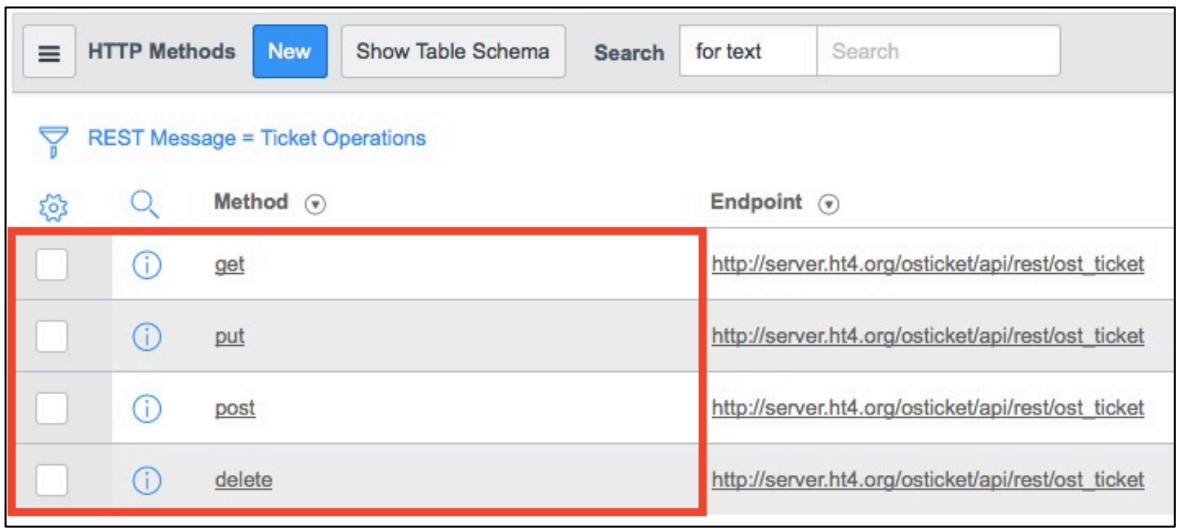
©2015 ServiceNow, Inc. All rights reserved.

3. Click **Submit**.

4. Once the new REST Message is listed, click **Ticket Operations** to open the record.



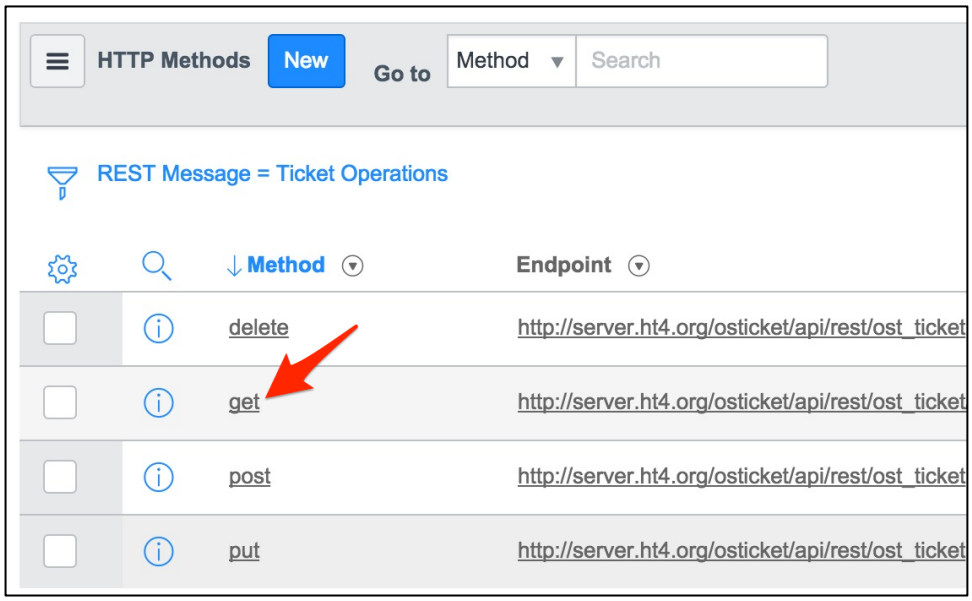
5. Confirm that the four basic HTTP operations are listed as **HTTP Methods.**



®2015 ServiceNow, Inc. All rights reserved.

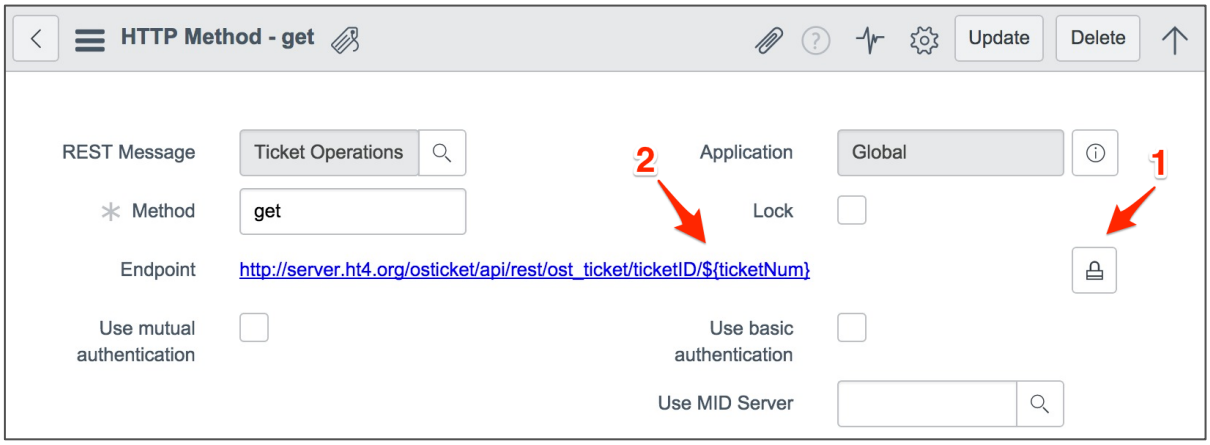
Set Up a Basic Ticket Query

1. Open the **get** HTTP Method inside of the **Ticket Operations** Rest Message record.



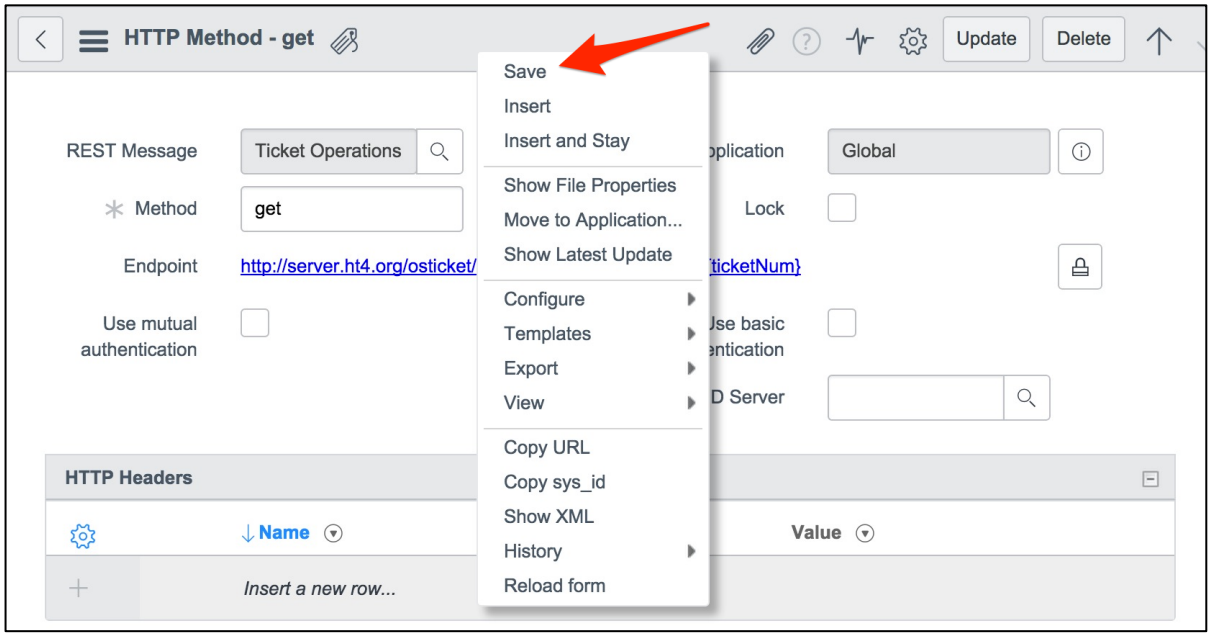
2. The API specification with this web service uses an endpoint with a suffix of: **ticketID/TICKET_NUM**. Since TICKET_NUM changes depending on which ticket you create, use a variable for it in the endpoint.

3. Modify the **REST Endpoint** to handle a variable-based ticket query by adding the following path to the current REST Endpoint: **/ticketID/${ticketNum}**. To modify the REST endpoint field, click the padlock next to the label.

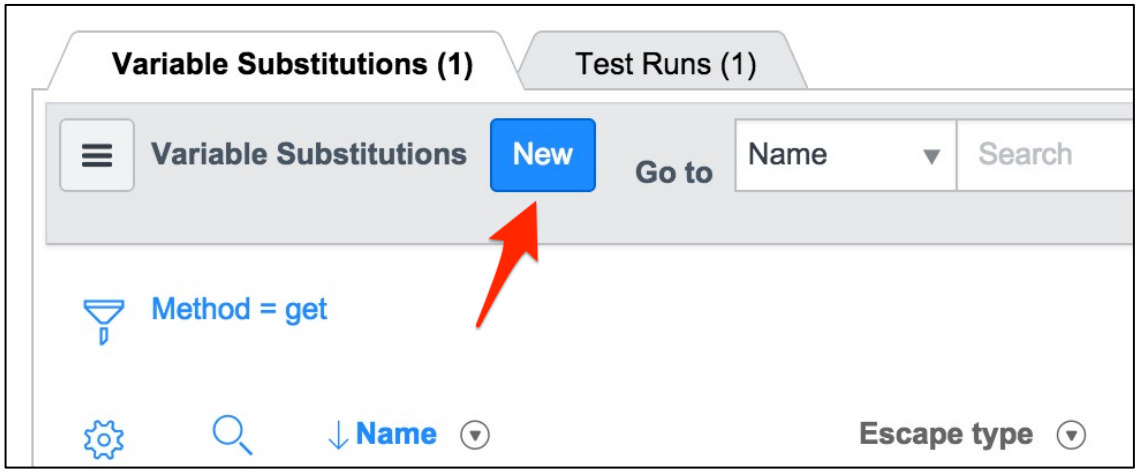


©2015 ServiceNow, Inc. All rights reserved.

4. Right-click the form header and **Save**.



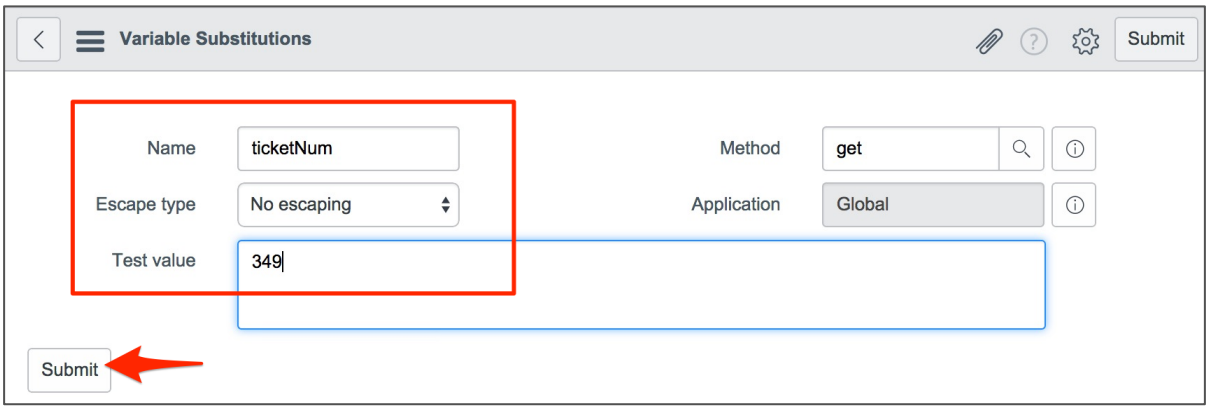
5. Create a test value for the variable you created in the endpoint to run sample requests. In the **Variable Substitutions** section of the form, click **New**.



©2015 ServiceNow, Inc. All rights reserved.

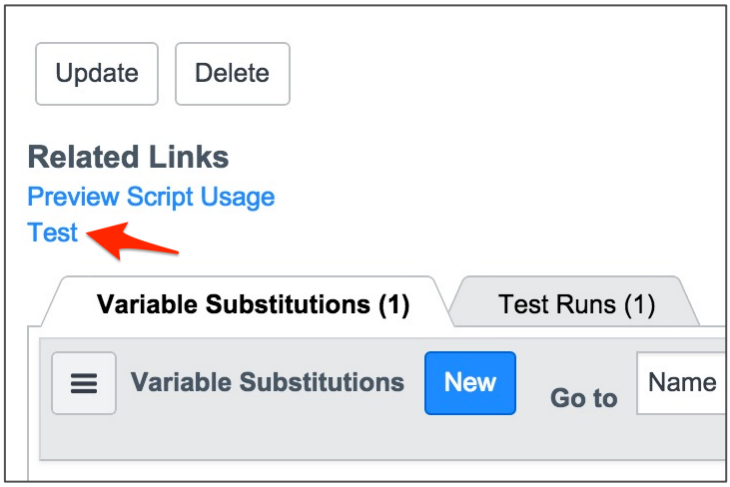
6. Complete the form as follows:

   Name: **ticketNum**
   Escape type: **No escaping**
   Value: **349**



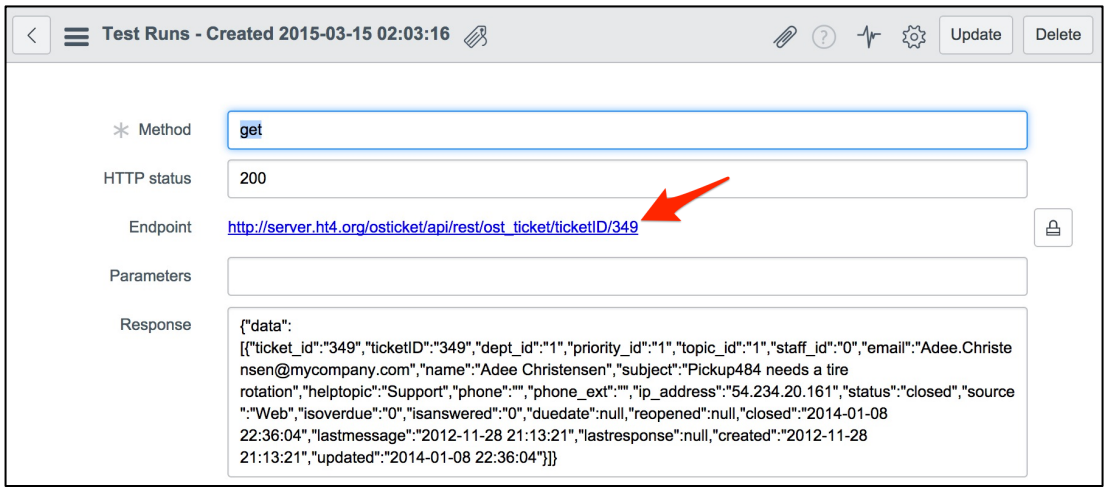
7. Click **Submit**.

8. Now that you have a sample variable, click the **Test** link on the HTTP Method form.



©2015 ServiceNow, Inc. All rights reserved.

9. You are redirected to a Test Run form that contains the Response details. Notice how the variable **${ticketNum}** in the Endpoint was replaced by the sample value you specified.



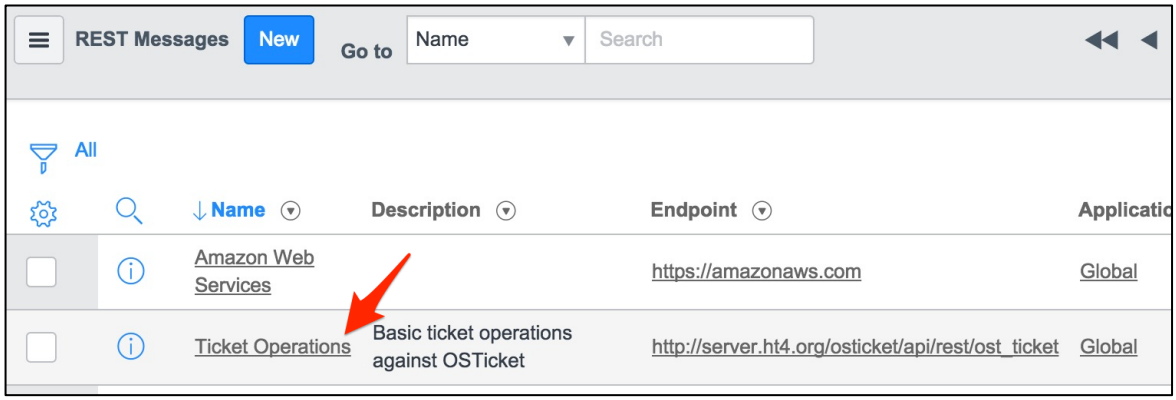
10. The **Response** field contains the JSON string that represents the data within the ticket in the OSTicket system. If you were to format that string, it would look like the following example.

```
"data":[
        {
          "ticket_id":"349",
          "ticketID":"349",
          "dept_id":"1",
          "priority_id":"1",
          "topic_id":"1",
          "staff_id":"0",
          "email":"Adee.Christensen@mycompany.com",
          "name":"Adee Christensen",
          "subject":"Pickup484 needs a tire rotation",
          "helptopic":"Support",
          "phone":"",
          "phone_ext":"",
          "ip_address":"54.234.20.161",
          "status":"closed",
          "source":"Web",
          "isoverdue":"0",
          "isanswered":"0",
          "duedate":null,
          "reopened":null,
          "closed":"2014-01-08 22:36:04",
          "lastmessage":"2012-11-28 21:13:21",
          "lastresponse":null,
          "created":"2012-11-28 21:13:21",
          "updated":"2014-01-08 22:36:04"
        }
]
```

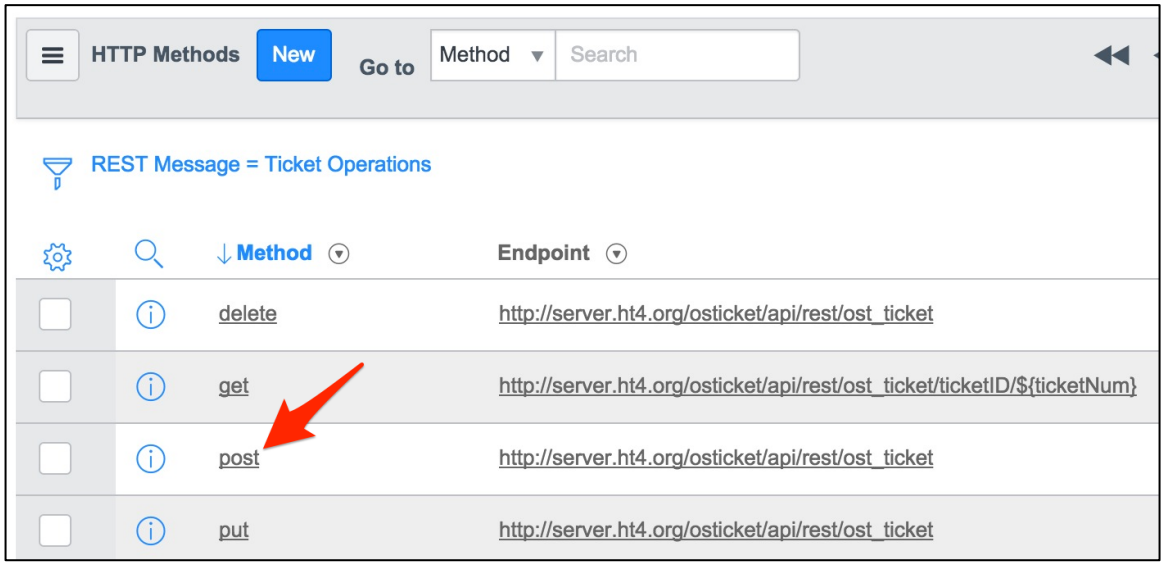
©2015 ServiceNow, Inc. All rights reserved.

Set Up a REST Request to Create a Ticket

1. Navigate to System **Web Services > Outbound > REST Message** and open the **Ticket Operations** record.



2. The OSTicket REST API requires an **HTTP POST** method to be used to create tickets. To set this up, click on the **post** HTTP Method.



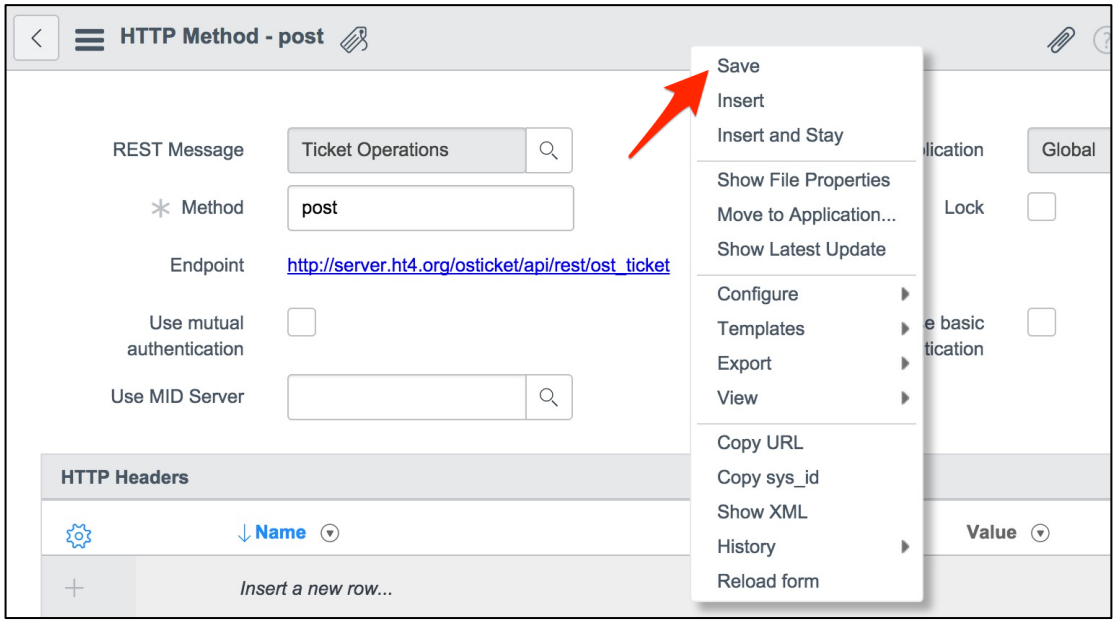
©2015 ServiceNow, Inc. All rights reserved.

3. The OSTicket API requires a JSON formatted string in the post content that contains field / value pairs. To do this, the following string needs to be entered in the **content** field of the **post** REST Message Function.



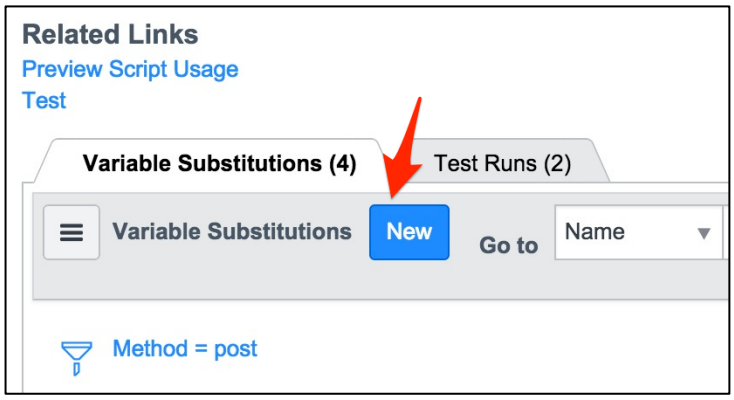
4. Notice that you inserted some variable strings in the content body so that you can insert data of your choice during each transaction.

5. Right-click the form header and **Save**.



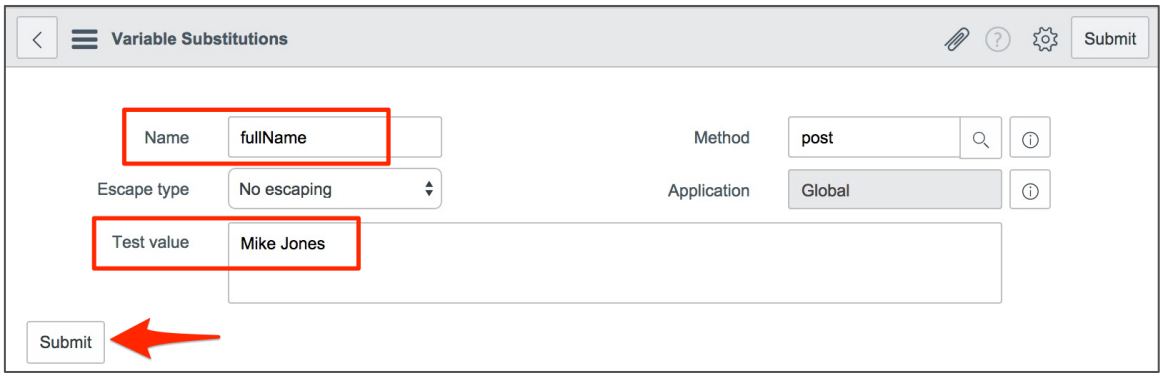
©2015 ServiceNow, Inc. All rights reserved.

6. Now create test variables for each variable in the JSON string. In the **Variable Substitutions** related list, click **New**.



7. Complete the form as follows:

   Variable Substitution 1
   Name: **fullName**
   Value: ***<your full name>***



8. Click **Submit**.

©2015 ServiceNow, Inc. All rights reserved.

9. In the Variable Substitutions related list, repeat the above two steps using the following data:

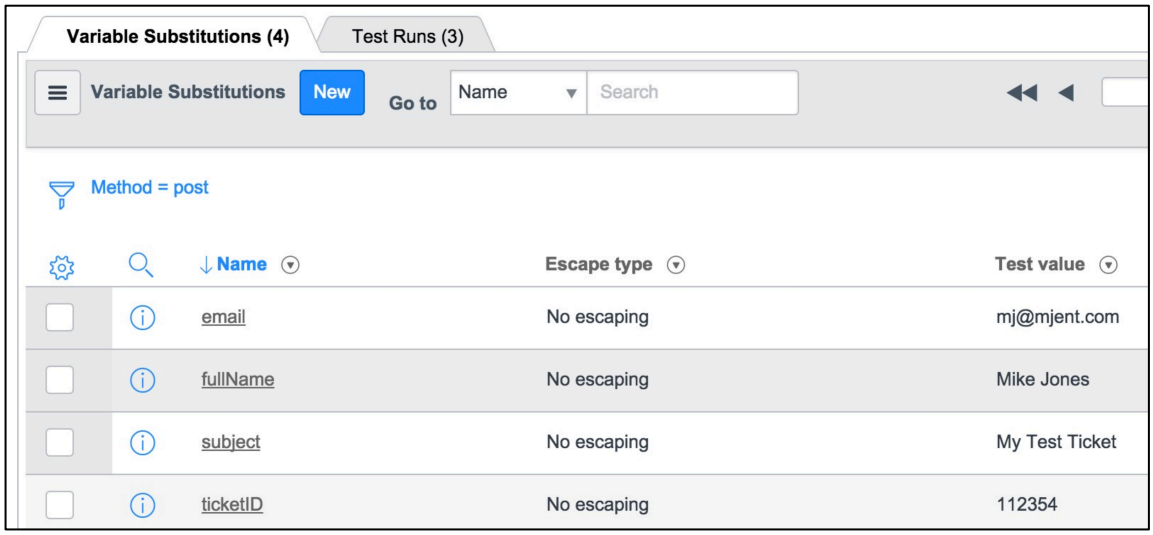
   Variable Substitution 2
   Name: **email**
   Value: ***<your email address>***

   Variable Substitution 3
   Name: **subject**
   Value: **My Test Ticket**

   Variable Substitution 4
   Name: **ticketID**
   Value: ***<a random integer value>*** (make note of this for later verification)

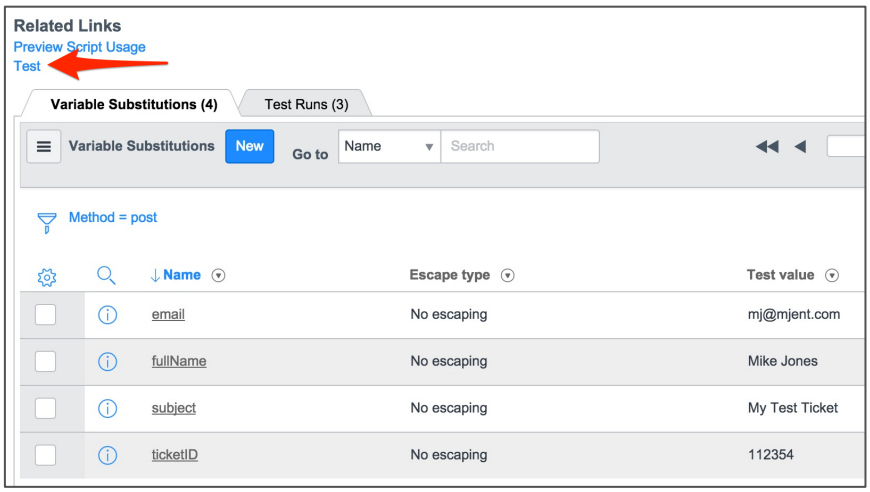
Note: Please note down the email address and ticketID value that you used in this step as they will be needed in subsequent steps. Ensure that the email address is unique.

10. The Variable Substitutions related list should now be populated as follows:

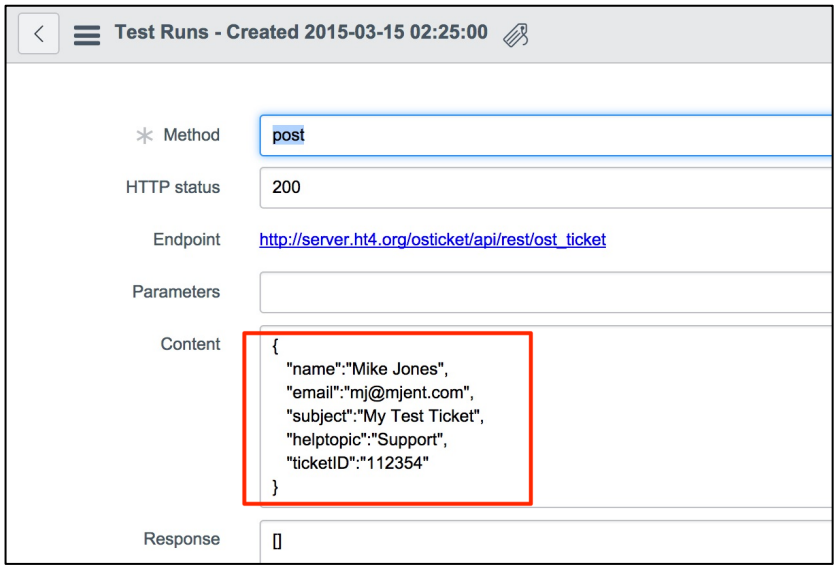


©2015 ServiceNow, Inc. All rights reserved.

11. Under Related Links, click **Test** to submit the information.



12. When the response page loads, notice that the **Content** field contains the actual body content that was posted to the server. The variables have been replaced with the values set in the previous few steps.

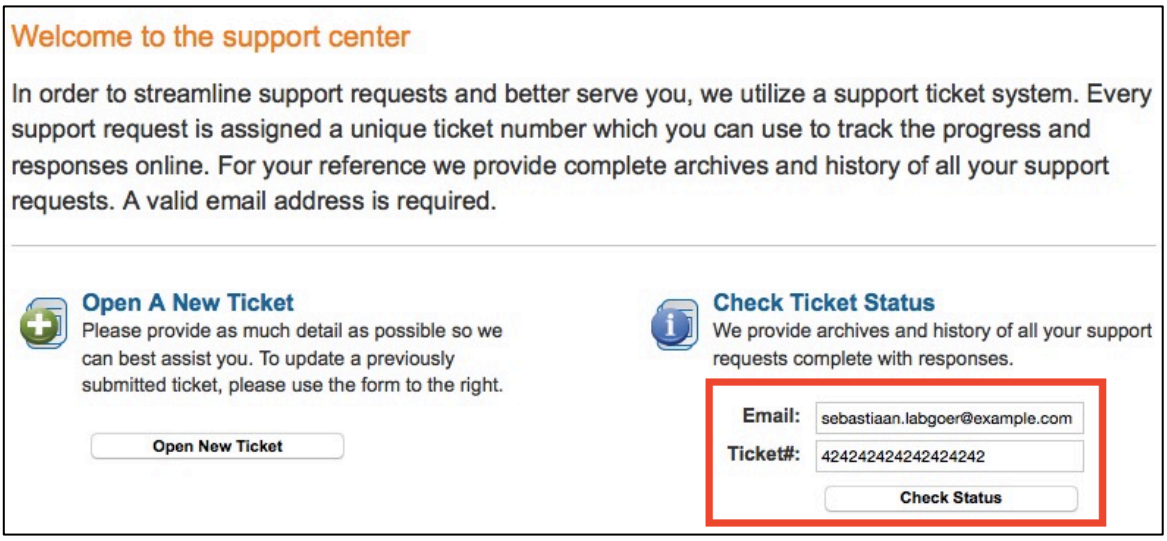


13. Unfortunately, this API does not return a descriptive response. Check the **OSTicket** user interface to confirm that the ticket was created.

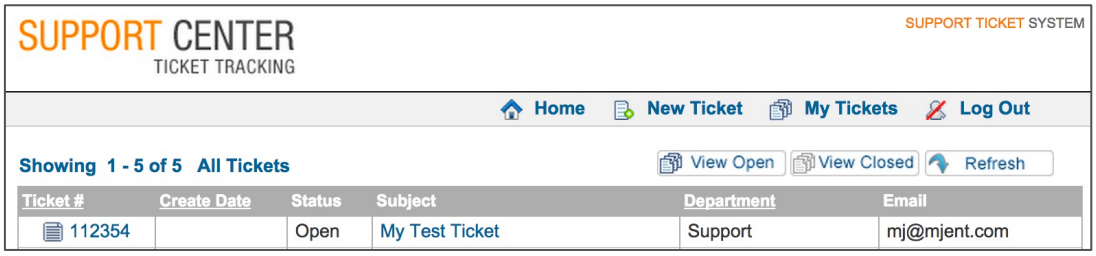
14. Open a new browser tab or window and go to the following URL: **http://server.ht4.org/osticket/**.

©2015 ServiceNow, Inc. All rights reserved.

15. In the login box, enter the email address used in the REST request as well as the **ticketID** variable .



16. The ticket should be listed in the system.



17. Click **Log Out**.

Report Your Completion of This Lab Exercise

1. Navigate to **Lab Management > Report Lab Progress**.
2. Enter your progress status.

©2015 ServiceNow, Inc. All rights reserved.

Lab Goal

Lab 4 Scripted REST Message

This lab explains how to leverage REST Message calls within ServiceNow scripts to provide dynamic communication between systems.

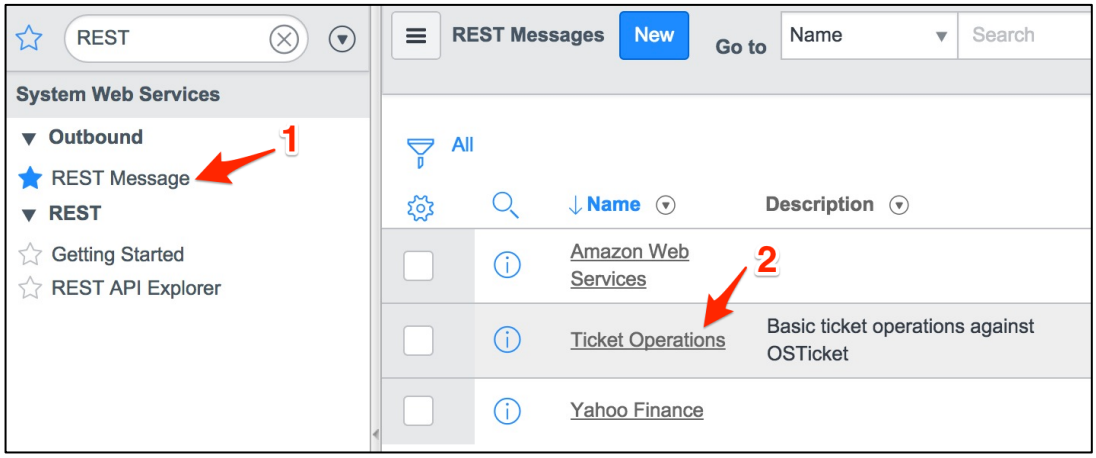
In this exercise, you leverage REST Message functions from within a business rule to create tickets in a third party ticketing system.

NOTE: Please see the 'Integration Lab 201 Knowledge' article in the 'Integration Lab 201' application for lab scripts and data that you can copy/paste in relevant steps to save you having to type these in manually.

Generate a Sample REST Message Function Script

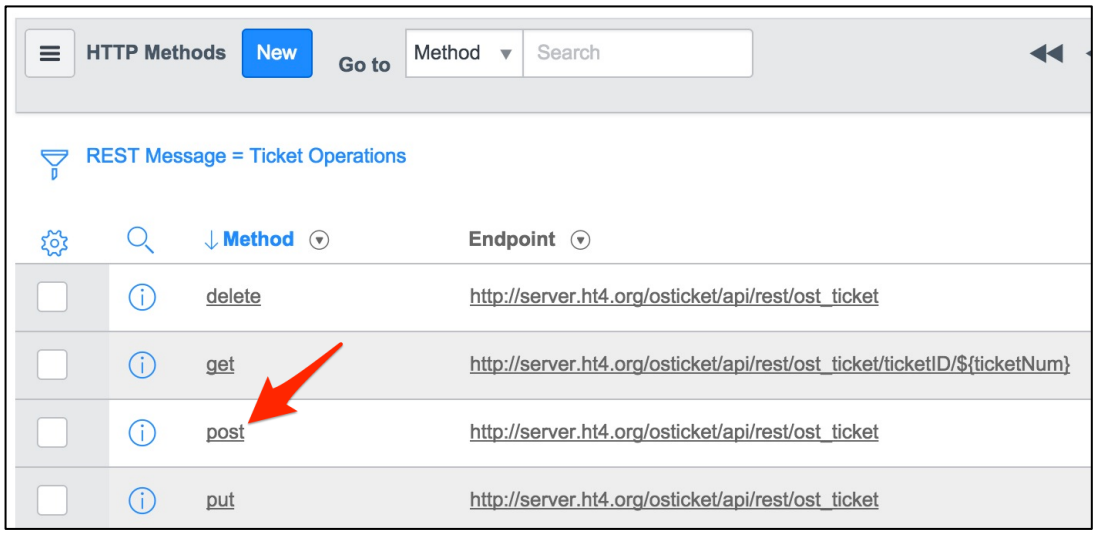
1. The REST Message Function tool auto generates a sample REST Message script based on function variables defined in the REST Message Function Parameters list.

2. Navigate to **System Web Services > Outbound > REST Message** and open the Ticket **Operations** record.



©2015 ServiceNow, Inc. All rights reserved.

3. In this exercise, you create a ticket in the third party system. You need to leverage the **post fu**nction to do this. In the **HTTP Methods re**lated list, click **post.**

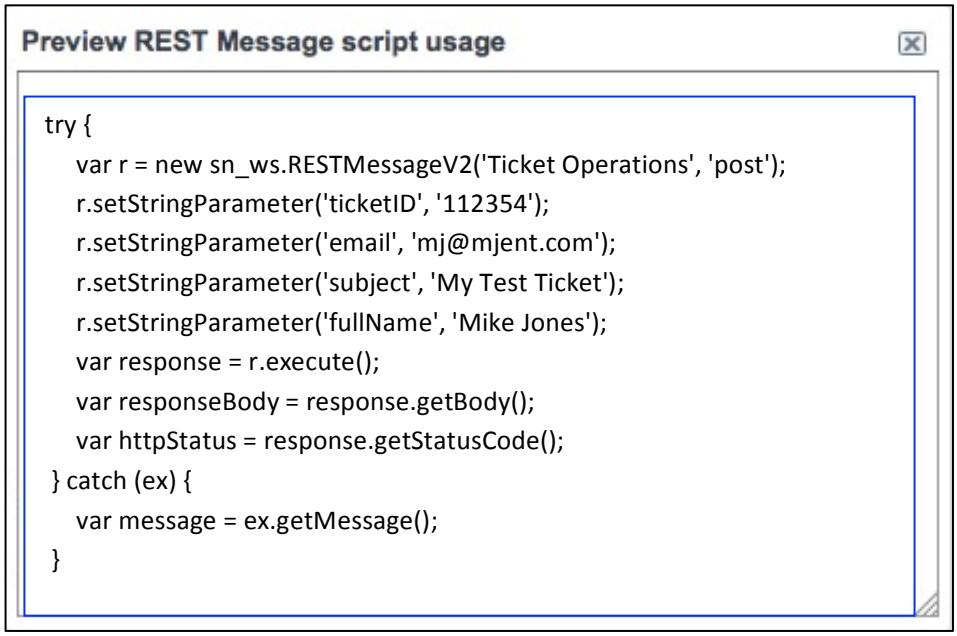


4. Since you already defined the sample REST Message Function Parameters, click the **Preview script usage** related link on the form.



©2015 ServiceNow, Inc. All rights reserved.

5. Copy the script to your clipboard for future use.

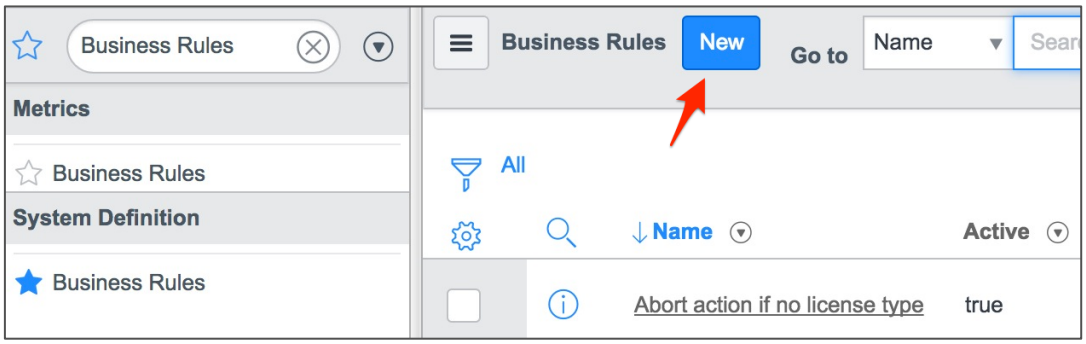


6. Close the Preview REST Message script usage dialog box.

Create a Business Rule on the Incident Table

When a new incident is created, you want to create a corresponding incident in the OSTicket system.

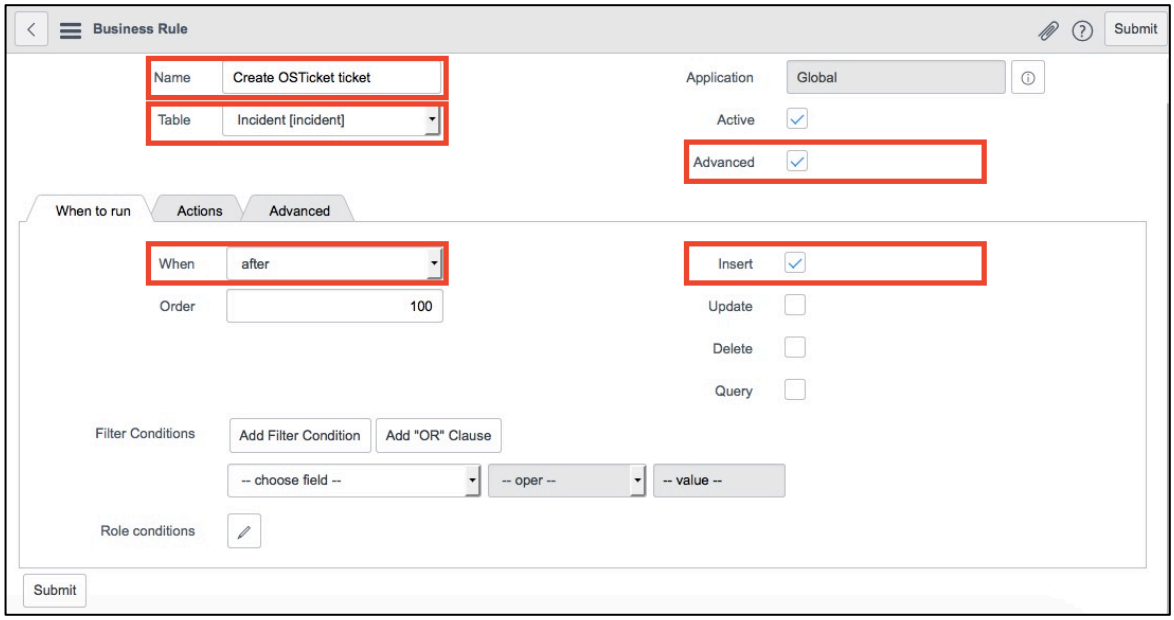
1. Navigate to **System Definition > Business Rules** and click **New**.



©2015 ServiceNow, Inc. All rights reserved.

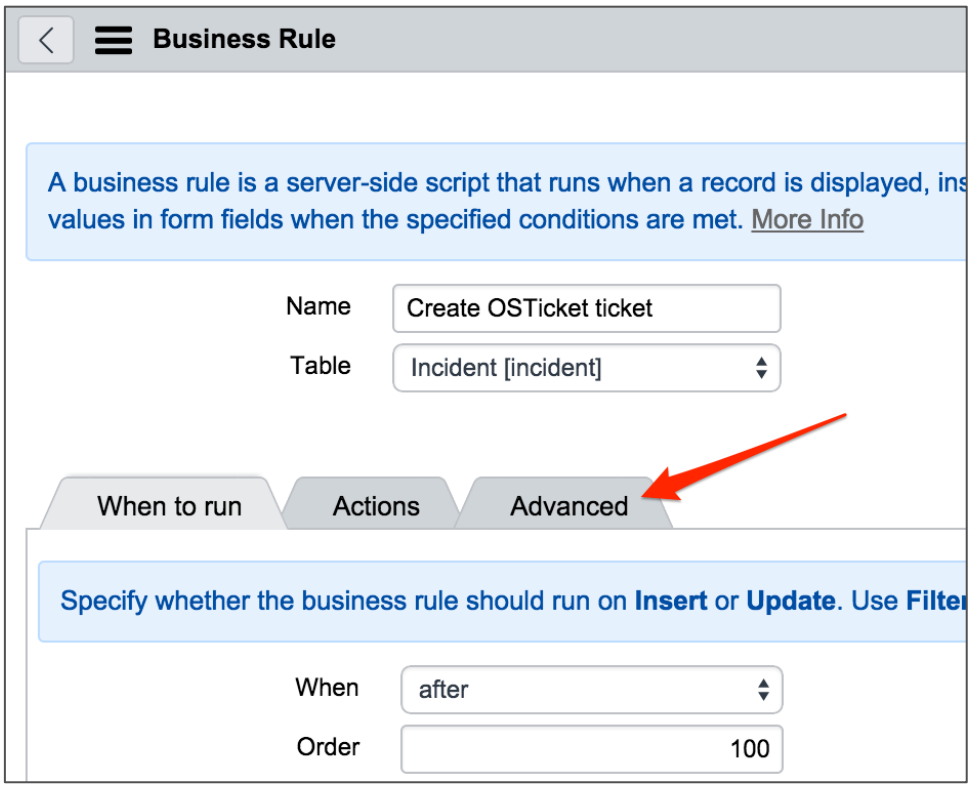
2. Complete the fields with the following information:

   Name: **Create OSTicket ticket**
   Table: **Incident [incident]**
   Advanced: **Checked**
   When: **after**
   Insert**: Checked**



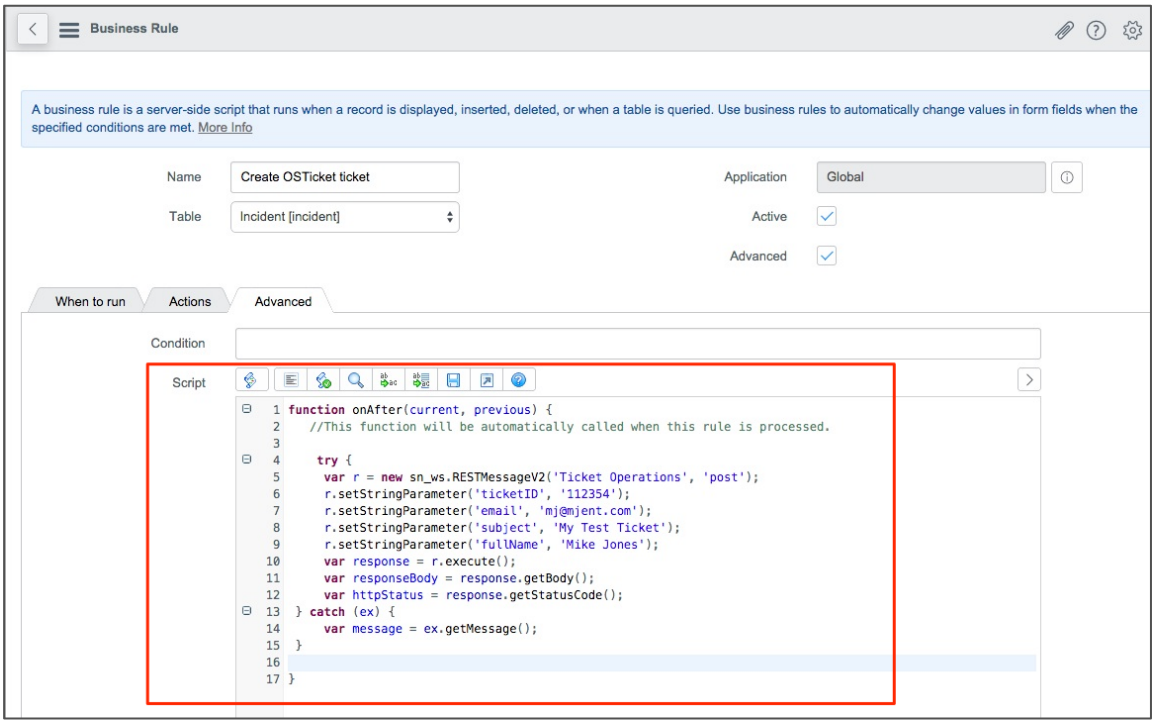
®2015 ServiceNow, Inc. All rights reserved.

3. Click the **Advanced tab** on the business rule form.



©2015 ServiceNow, Inc. All rights reserved.

4. In the **Actions** form section, paste the auto-generated code from the REST Message Function into the Script field.



5. Modify the script so that it uses **Incident** values rather than the hard coded strings you used in testing the function. Use the following Incident fields to set the REST variables in the script:

   subject: **current.short_description**
   fullName: **current.caller_id.name**
   ticketID: **current.number**

©2015 ServiceNow, Inc. All rights reserved.

Note: Please ensure your script reflects the email address you used previously.

```
function onAfter(current, previous) {
          try {
               var r = new sn_ws.RESTMessageV2('Ticket Operations', 'post');
               r.setStringParameter('ticketID', current.number);
               r.setStringParameter('email', 'mj@mjent.com');
               r.setStringParameter('subject', current.short_description);
               r.setStringParameter('fullName', current.caller_id.name);
               var response = r.execute();
               var responseBody = response.getBody();
               var httpStatus = response.getStatusCode();
          } catch(ex) {
                    var message = ex.getMessage();
          }
}
```

Note: the email address is still hardcoded. This allows you to log into the OSTicket system later with the email address you originally used to confirm ticket creation in a later step.

6. Click **Submit** to save the business rule.

Test This Portion of the Integration

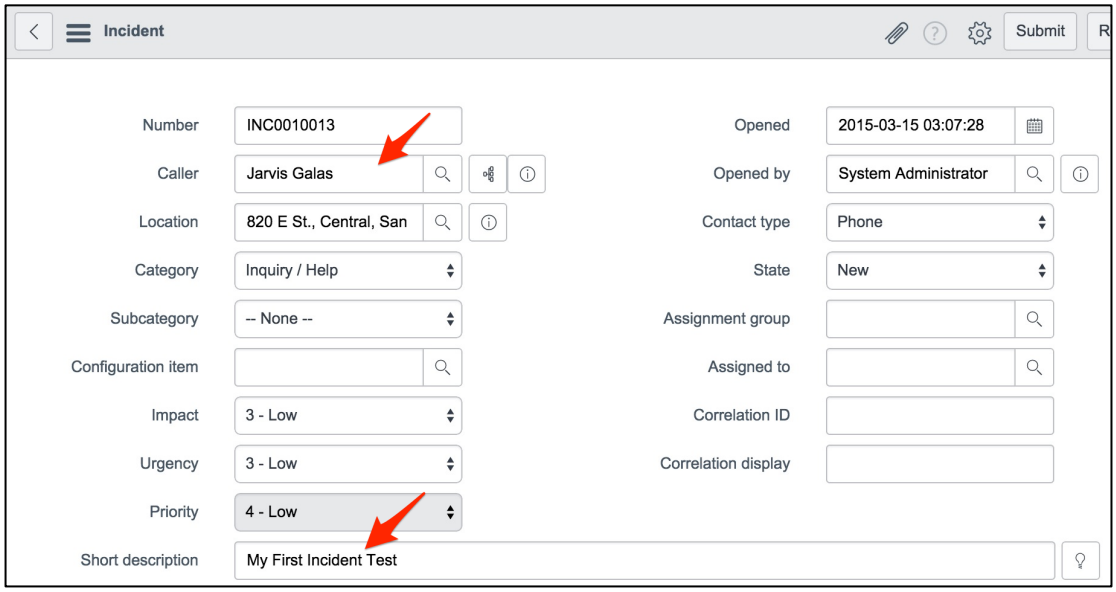
Create an incident to confirm that whatever incident is created in ServiceNow also gets created in the OSTicket system.

1. Navigate to **Incident > Create New**.

2. Set the Caller to **Jarvis Galas**.

©2015 ServiceNow, Inc. All rights reserved.

3. Set the Short Description to **My First Incident Test**.



4. Copy the Incident **Number** to the clipboard for a future step.

5. Click **Submit**.

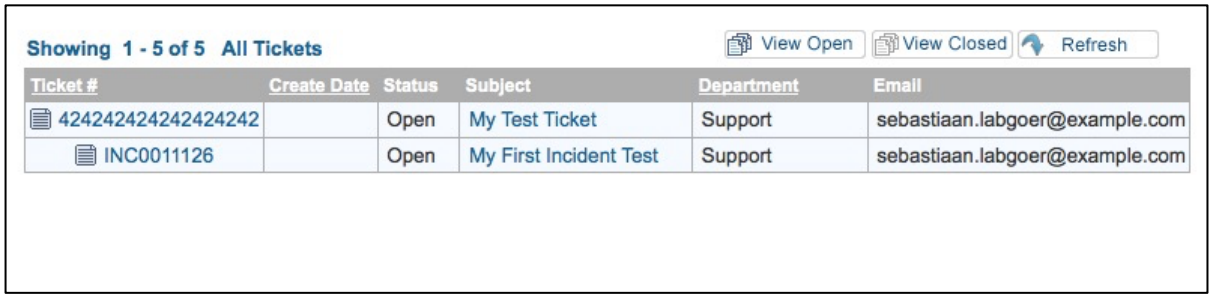
6. Open a new browser tab or window and go to the following URL: **http://server.ht4.org/osticket/**.

7. In the login box, enter the email address you used the business rule and the Incident Number that you copied from the ServiceNow incident record. Click **Check Status**.



©2015 ServiceNow, Inc. All rights reserved.

8. The ticket should be listed in the system.



Note: **If for any reason the login fails**, use the email address and ticket number created in Lab 3. The newly created ticket should then appear in the list. This can occur due to a quirk in the example web service we are using for the lab.

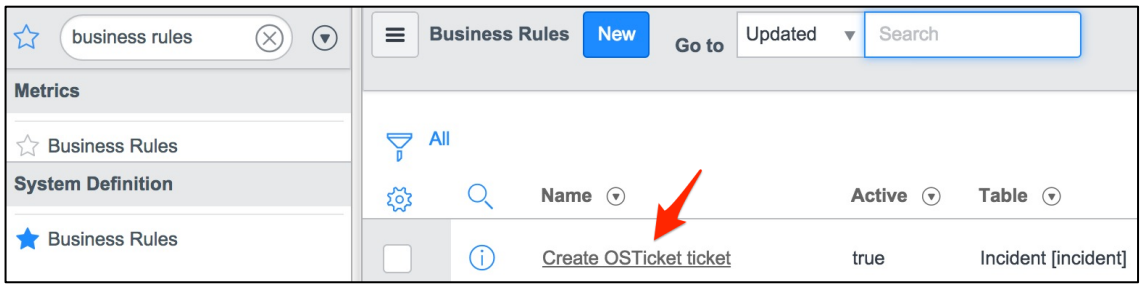
9. Click **Log Out**.

®2015 ServiceNow, Inc. All rights reserved.

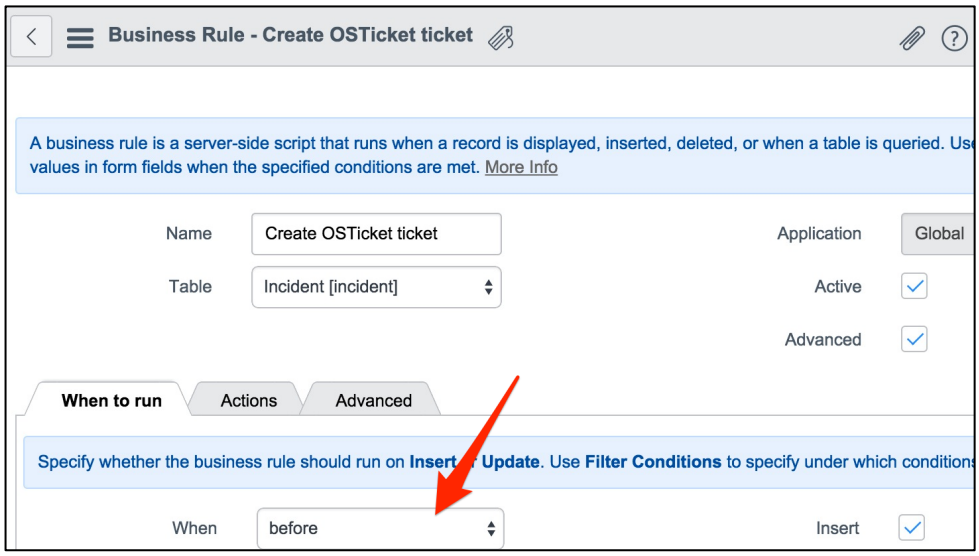
Set Up the Business Rule to Store the OSTicket GUID for the Ticket

You now want to store the internal Globally Unique Identifier (GUID) number of the OSTicket that was created in the third-party system so that you can communicate with OSTicket in the future. OSTicket uses a **ticket_id** field for its unique GUID. The similar **ticketID** field in the OSTicket system is the display value of the ticket number.

1. Navigate to **System Definition > Business Rules** and open the Create OSTicket ticket record.



2. You want to obtain the GUID from the external system and save it with the incident record in ServiceNow. Instead of having the Business Rule run **after** the data is saved, set the **When to run** value to **before**. Note: Typically integrations are carried out using **after** as part of the condition for the business rule, we are using **before** here for demonstration purposes.



©2015 ServiceNow, Inc. All rights reserved.

3. Append the following code to the **Script** field in the Advanced tab directly after the line containing 'var httpStatus = response.getStatusCode();'. This code calls the **get REST** Message Function to get details on the incident that was just created.

```
//Make a query request for the ticket that was just created
var details = new sn_ws.RESTMessageV2('Ticket Operations', 'get');
details.setStringParameter('ticketNum', current.number);
var detailsResponse = details.execute();
```

4. You will then need to get the response body string from the response object.

```
//Get the response string
var jsonString = detailsResponse.getBody();
```

5. You may recall the response from the OSTicket system will look like the following JSON String.

```
"data":[
       {
         "ticket_id":"349",
         "ticketID":"349",
         "dept_id":"1",
         "priority_id":"1",
         "topic_id":"1",
         "staff_id":"0",
         "email":"Adee.Christensen@mycompany.com",
         "name":"Adee Christensen",
         "subject":"Pickup484 needs a tire rotation",
         "helptopic":"Support",
         "phone":"",
         "phone_ext":"",
         "ip_address":"54.234.20.161",
         "status":"closed",
         "source":"Web",
         "isoverdue":"0",
         "isanswered":"0",
         "duedate":null,
         "reopened":null,
         "closed":"2014-01-08 22:36:04",
         "lastmessage":"2012-11-28 21:13:21",
         "lastresponse":null,
         "created":"2012-11-28 21:13:21",
         "updated":"2014-01-08 22:36:04"
       }
]
```

6. You need to convert this string into a JavaScript object with the following code appended in the **Script** field.

```
//Convert the JSON String to an object
var parser = new JSONParser();
var parsed = parser.parse(jsonString);
```

©2015 ServiceNow, Inc. All rights reserved.

7. The resulting object has the following structure:

```
parsed (object)
   data (array)
      name (string)
      dept_id (string)
      ticket_id (string)
      …etc....
```

8. When performing an integration, it is a common practice to obtain the third-party's record GUID to save with the ServiceNow record as well as a tag that indicates with which system the record is integrated. There are two fields on task tables for this purpose.

   correlation_id: this field typically holds the third-party GUID of the associated record

   correlation_display: this field is usually a hard coded string that indicates which third-party system is integrated with this task

Note: If these fields are not present on the incident form on your instance please enable these fields.

9. Append the following code to the **Script** to set the ticket_id GUID to the incident's **correlation_id** field as well as to set the **correlation_display** field to be **OSTicket:**

```
current.correlation_id = parsed.data[0].ticket_id;
current.correlation_display = "OSTicket";
```

©2015 ServiceNow, Inc. All rights reserved.

10. The business rule **Script** should now look similar to the following example (email address should be different).

```
function onBefore(current, previous) {
    try {
        var r = new sn_ws.RESTMessageV2('Ticket Operations', 'post');
        r.setStringParameter('ticketID', current.number);
        r.setStringParameter('email', 'mj@mjent.com');
        r.setStringParameter('subject', current.short_description);
        r.setStringParameter('fullName', current.caller_id.name);
        var response = r.execute();
        var responseBody = response.getBody();
        var httpStatus = response.getStatusCode();

        //Make a query request for the ticket that was just created
        var details = new sn_ws.RESTMessageV2('Ticket Operations', 'get');
        details.setStringParameter('ticketNum', current.number);
        var detailsResponse = details.execute();

        //Get the response string
        var jsonString = detailsResponse.getBody();

        //Convert the JSON String to an object
        var parser = new JSONParser();
        var parsed = parser.parse(jsonString);
        current.correlation_id = parsed.data[0].ticket_id;
        current.correlation_display = "OSTicket";
    } catch(ex) {
        var message = ex.getMessage();
    }
}
```

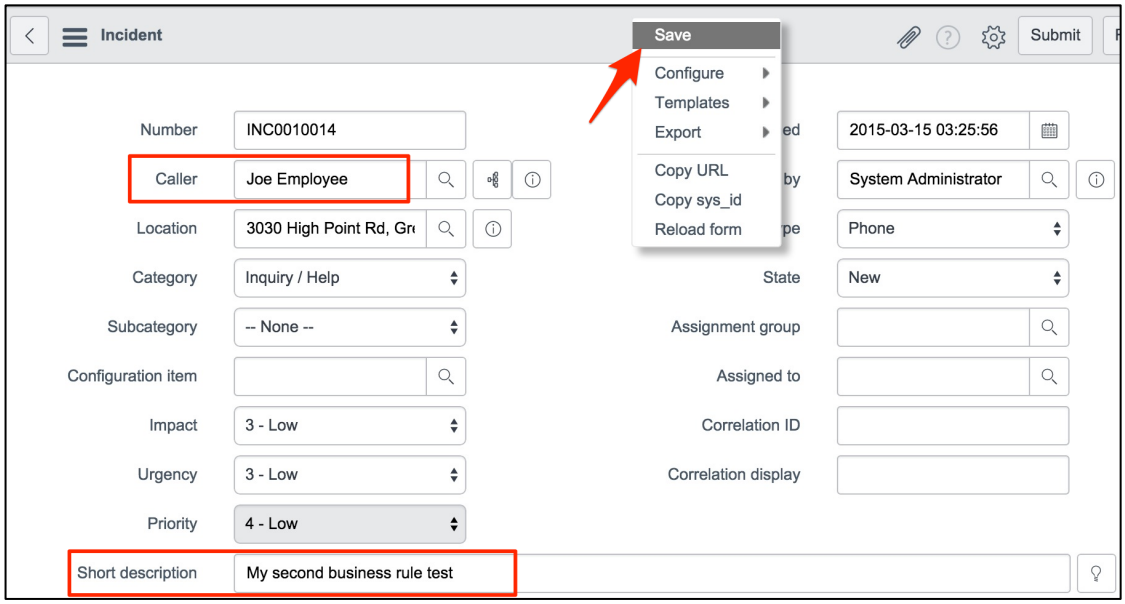
11. Click **Update**.

Test the Integration

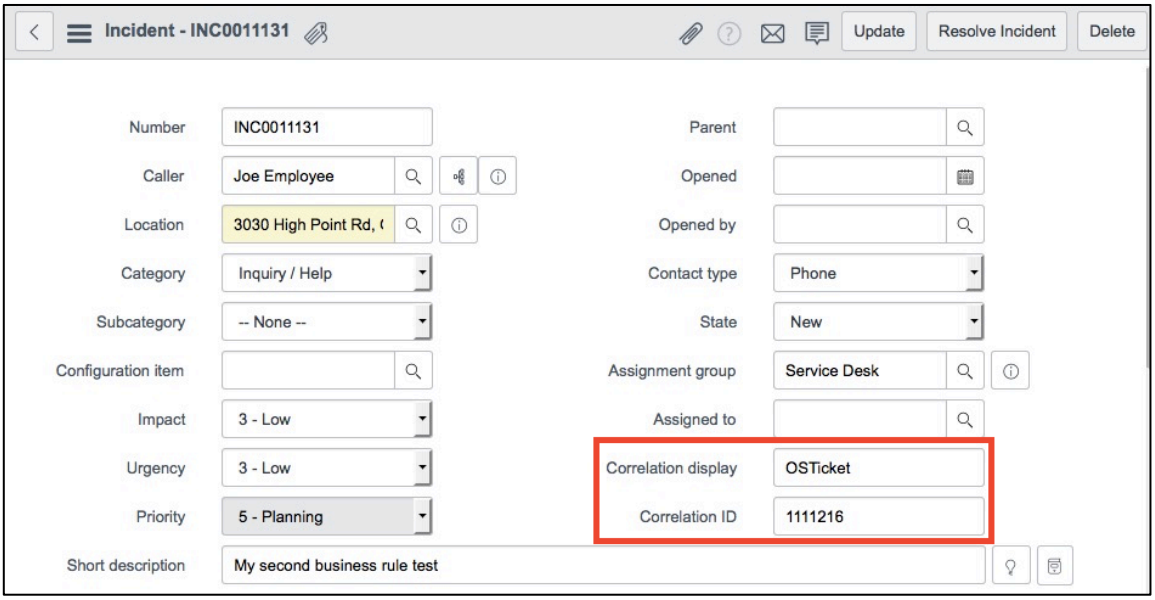
1. Navigate **to Incident > Create New**.
2. Set Caller to **Joe Employee**.
3. Set **Short Description** to **My second business rule test**.

©2015 ServiceNow, Inc. All rights reserved.

4. Right-click the form header and **Save**.



5. After the form refreshes, notice the values of the **Correlation display** and **Correlation ID** fields. They should look similar to the example, but have a different ID.



®2015 ServiceNow, Inc. All rights reserved.

Report Your Completion of This Lab Exercise

1. Navigate to **Lab Management > Report Lab Progress**.
2. Enter your progress status.

®2015 ServiceNow, Inc. All rights reserved.

Lab Goal

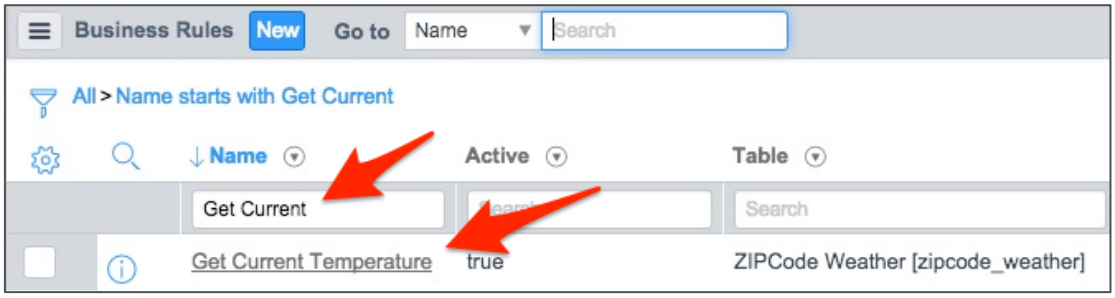
Optional Lab 5 ECC Queue Retry Policy

This lab explains how to submit and handle Asynchronous SOAP requests. Additionally, you learn how to create and leverage ECC Queue Retry Policies to make your web service calls more robust.

In this exercise, you replace the business rule created in the previous exercise on the **ZIPCode Weather** table. This new business rule updates the weather for a ZIP Code asynchronously when a ZIP code is entered or changed. You then write an ECC Queue Retry policy and simulate a failure scenario for the web service calls. You confirm the Retry Policies are firing and then you turn off the failure simulation and ensure the retries succeed in getting a response.

Disable the Prior Business Rule

1. Navigate to **System Definition > Business Rules** and open the business rule: **Get Current Temperature**.

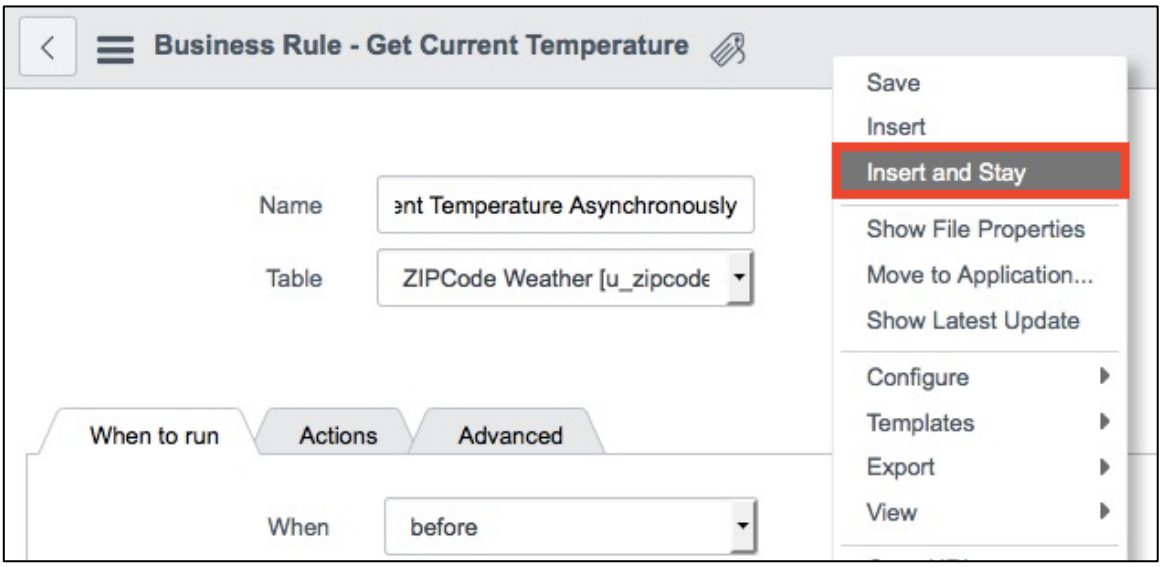


2. Deactivate the rule by deselecting the **Active** check box, then right-click the title bar and **Save**.

©2015 ServiceNow, Inc. All rights reserved.

Create a New Business Rule to Run Asynchronously

1. Right-click on the title bar of the business rule record just deactivated and click **Insert and Stay**.



2. Once the form refreshes from the insert, rename the business rule to be: **Get Current Temperature Asynchronously**.

3. Set the **Active** field to be checked.

4. Change the **When** field to be: **async**. This executes the business rule whenever there is a free worker process available rather than to hold the User interface until the call is complete.

©2015 ServiceNow, Inc. All rights reserved.

5. Change the script to the following:

```
function onAsync(current) {
    //This function will be automatically called when this rule is processed.
    try {
        var s = new sn_ws.SOAPMessageV2('Lab Weather - My Weather',
'WeatherSoap12.GetCityWeatherByZIP');
        s.setStringParameter('ZIP', current.zipcode);
        var response = s.executeAsync();
        response.waitForResponse(60);
        if (response.haveError()) {
            throw gs.getMessage("Problem with response");
        }
        var xmldoc = new XMLDocument(response.getBody()); //Convert the XML
response string to an XMLDocument object
        temperature = xmldoc.getNodeText("//Temperature");
        if (JSUtil.nil(temperature)) {
            throw gs.getMessage("Failed to acquire valid temperature");
        }
    } catch (ex) {
        temperature = "-273.15";
        gs.info(ex);
    } finally {
        current.temperature = temperature;
        current.update();
    }
};
```

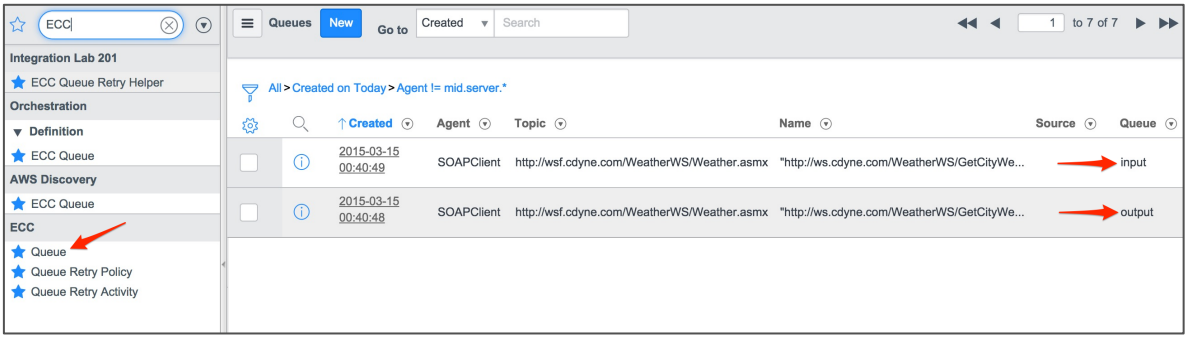
6. Right click the title bar of the business rule and click **Save**.

Confirm the business rule runs asynchronously

1. Navigate to Integration Lab 201 > ZIPCode Weather.
2. Click **New.**
3. Set the **ZIPCode** to **83404** and click **Submit**.
4. Notice that when the page reloads, there is no temperature listed. This is due to the business rule running asynchronously.
5. Update the list by clicking on **Integration Lab 201 > ZIPCode Weather** module again. Do this repeatedly until a temperature appears. (It should take no longer than 30 seconds.)

©2015 ServiceNow, Inc. All rights reserved.

Note: To see what happens on the ECC queue, navigate to **ECC > Queue.** Notice that for an Agent type of SOAPCLient there is an output entry on the queue (what we sent to the endpoint) and a corresponding input entry (the response from the endpoint).



Create an ECC Queue Retry Policy

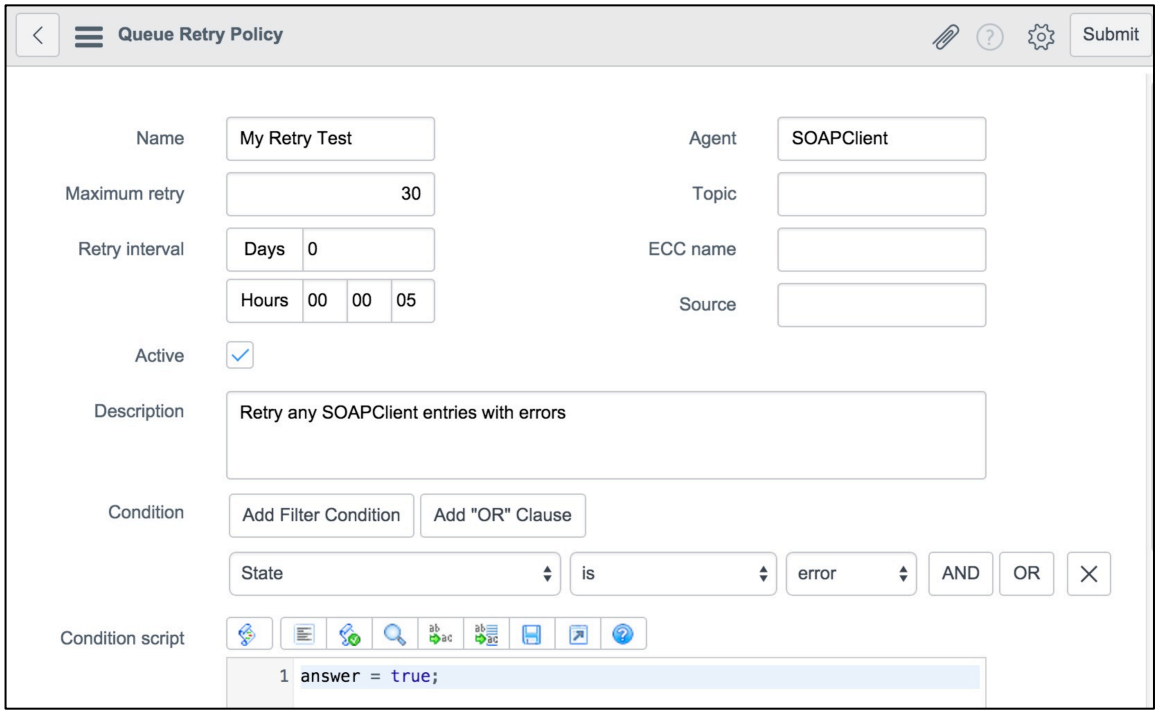
1. Browse to ECC > Queue Retry Policy.
2. Click **New**.
3. Enter the following information:`

Name	**My Retry Test**
Maximum Retry	**30**
Retry Interval	**0 Days 00 Hours 00 Minutes 05 Seconds**
Active	**True**
Agent	**SOAPClient**
Description	**Retry any SOAPClient entries with errors**
Condition	**State is error**

4. Set the Condition Script field to be: **answer = true**;

®2015 ServiceNow, Inc. All rights reserved.

5. The policy should look like the example.



6. Click **Submit**.

Test the Retry Policy

1. In order to show ECC Queue Retry policies, we have included a utility that causes all outbound asynchronous SOAP requests to fail.

2. Browse **to Integration Lab 201 > ECC Queue Retry Helper**.

3. Click the **Automatically Fail SOAP Requests** button.

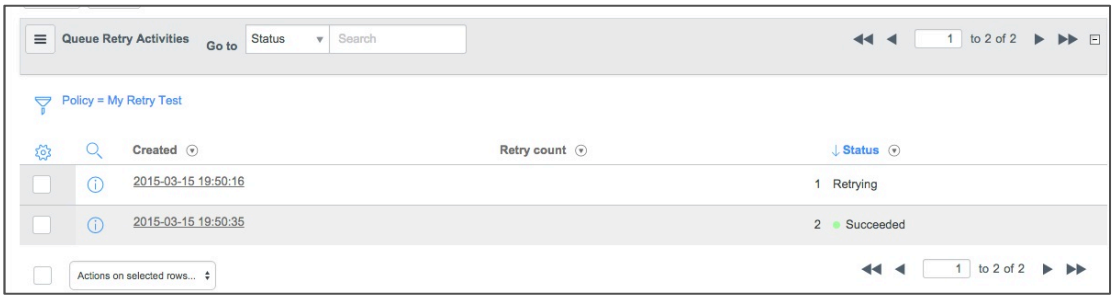
4. Submit a new SOAP request by browsing to the **Integration Lab 201 > ZIPCode Weather** module and creating a new record with the ZIP code of **83201**.

5. Wait for approximately **30** seconds.

6. Browse to the retry policy record by going to **ECC > Queue Retry Policy** and opening the **My Retry Test** policy.

©2015 ServiceNow, Inc. All rights reserved.

7. At the bottom of the record there is a **Queue Retry Activities** list or tab. It should list a few retry attempts.

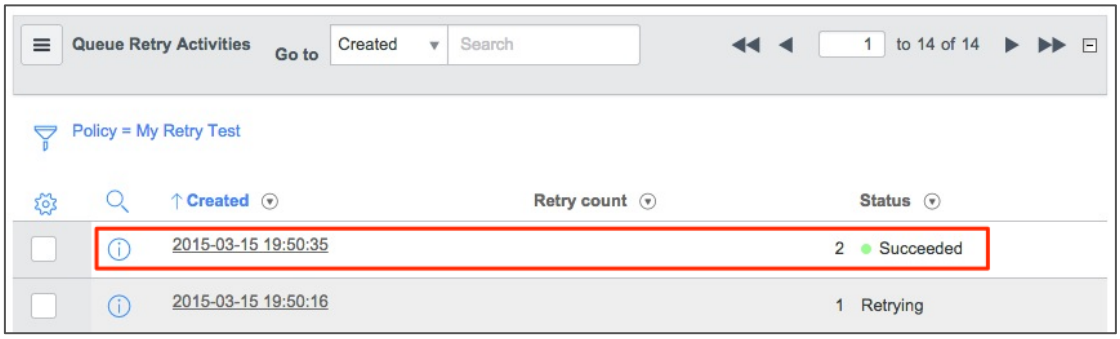


8. Go back to the **Integration Lab 201 > ECC Queue Retry Helper** module and click the **Allow SOAP Requests to Succeed** button.

9. Wait approximately **30** seconds for the last retry to fire and register in the system.

10. Browse back to the retry policy record by going to **ECC > Queue Retry Policy** and opening your **My Retry Test** policy.

11. At the bottom of the record, take another look at the **Queue Retry Activities** list or tab. It should list a few retry attempts,



Report Your Completion of This Lab Exercise

1. Navigate to **Lab Management > Report Lab Progress**.

2. Enter your progress status.

©2015 ServiceNow, Inc. All rights reserved.